

**INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN**

**Modelo para la Construcción de Algoritmos Apoyados en
Heurísticas**

Tesis que para obtener el grado de
DOCTOR EN CIENCIAS DE LA COMPUTACIÓN
PRESENTA

M.C. Ricardo Vargas de Bastera

Director de la Tesis: Dr. Agustín Gutiérrez Tornés
Codirector: Dr. Sergio Suárez Guerra

México, D. F.

Noviembre 2006



INSTITUTO POLITÉCNICO NACIONAL SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D. F. siendo las 15:30 horas del día 20 del mes de Octubre de 2006 se reunieron los miembros de la Comisión Revisora de Tesis designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis de grado titulada:

“MODELO PARA LA CONSTRUCCIÓN DE ALGORITMOS APOYADOS EN HEURÍSTICAS”

Presentada por el alumno:

VARGAS	DE BASTERRA	RICARDO
Apellido paterno	materno	nombre(s)
Con registro:		
B	0	2
0	9	2
7		

aspirante al grado de: **DOCTOR EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron **SU APROBACIÓN DE LA TESIS**, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Presidente

Secretario

DR. GRIGORI SIDOROV

DR. JESÚS MANUEL OLIVARES CEJA

Primer vocal
(Director de Tesis)

Segundo vocal
(Co-director)

DR. AGUSTÍN FRANCISCO GUTIÉRREZ-TORNÉS

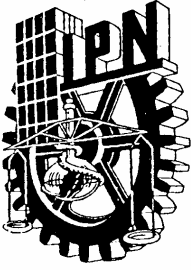
DR. SERGIO SUÁREZ GUERRA

Tercer vocal

DRA. AURORA PÉREZ ROJAS

EL PRESIDENTE DEL COLEGIO

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN
EN COMPUTACIÓN
DIRECCIÓN



INSTITUTO POLITECNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 20 del mes de octubre del año 2006, el (la) que suscribe VARGAS DE BASTERRA RICARDO alumno (a) del Programa de Doctorado en Ciencias de la Computación con número de registro B020927, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Agustín Gutiérrez Tornés y co-dirección de Dr. Sergio Suárez Guerra, cede los derechos del trabajo intitulado MODELO PARA LA CONSTRUCCIÓN DE ALGORITMOS APOYADOS EN HEURÍSTICAS, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección r_vargas@att.net.mx. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Ricardo Vargas de Basterra

Dedicatoria

A lo largo de los años distintas personas han estado a mi lado tanto en los momentos de alegría como en aquellos que lo ponen a uno a prueba. Todas ellas son Familia. Justamente a la Familia es a quien dedico este trabajo.

La Familia alimenta, protege, educa, forma, cura, impulsa, abriga y desarrolla al cuerpo y espíritu de las personas. Es el calor del hogar alimentado por profundo amor prodigado a todos y cada uno de sus miembros, los pequeños, los mayores, los ancianos, incluso los que ya se fueron de este mundo.

Sin lugar a dudas la Familia a la que pertenezco es la que ha permitido hacer este trabajo. El esfuerzo, las privaciones y las ausencias las ha soportado con gran espíritu de solidaridad, de pertenencia, de tolerancia y paciencia.

Querida Familia, hoy te dedico este trabajo, fruto del tiempo que les he robado, recíbanlo como un tributo al amor que nos ha unido.

Ricardo Vargas de Basterra

Mi Familia:

Eduardo Vargas Rosas[†]
Perla Aurora de Basterra de Vargas
Aurora Rosas de Escalante
Rogelio Escalante Navarro[†]
Alina Escalante Rosas
Enrique Vargas Basterra
Myrna Escalante Rosas
Ricardo Vargas Escalante
Samantha Vargas Escalante

Y varios mas que omito por abreviar pero que saben que están en mi corazón.

Agradecimientos

Muchas y distintas personas me han ayudado para poder contar el día de hoy con este trabajo, algunos con una acción muy directa y otros con interesantes conversaciones que me aportaron ideas diversas sobre como enfrentar el problema planteado. A todas ellas mi más profundo agradecimiento.

Hacer una lista siempre conlleva el riesgo de olvidar a alguien, por favor, cualquier omisión en estos agradecimientos se debe sin duda alguna a un error involuntario.

Quisiera iniciar agradeciendo a mi director de tesis, Dr. Agustín Gutiérrez Tornés, por la paciencia y confianza que me tuvo al considerarme digno de trabajar con él en este proyecto. A mi codirector, Dr. Sergio Suárez Guerra, por su incondicional apoyo, palabras de aliento y sabios consejos.

Debo agradecer a mi sínodo formado, además de mi director y codirector de tesis, por la Dra. Aurora Pérez Rojas, el Dr. Grigorí Sidorov y el Dr. Jesús Manuel Olivares Ceja, quienes revisaron este trabajo y siempre aportaron recomendaciones positivas y buenos consejos sobre como enriquecer y mejorar en todos sus aspectos esta tesis.

Quiero también agradecer al resto de los compañeros del Instituto Politécnico Nacional, en particular a los miembros del Centro de Investigación en Computación por su compañerismo y profesionalismo en cada una de las cosas que emprenden.

También quiero agradecer a mis compañeros de trabajo de la Universidad del Valle de México, por sus comentarios de aliento e incondicional apoyo manifestado de muy diversas maneras, entre todos ellos están el Ing. Jesús Alfonso Marín Jiménez, quien es un mentor natural; al Dr. Eduardo García García, por su amistad; al Mtro. Luís Silva Guerrero, por las facilidades otorgadas para hacer este trabajo; y a mis colaboradores que son verdaderos amigos de tiempo completo, Verónica Cruz, Julio Cesar Morales, Rocío Chávez, Carmen Cerón y Rosario Ortega quienes me han tolerado con gran paciencia y comprensión.

No puedo dejar de agradecer a los grupos de estudiantes que han participado activamente en el proceso de esta tesis sufriendo en carne propia las ideas, principios y teorías que ahora se integran en este trabajo.

Para finalizar, también quiero darle las gracias por muy diversos motivos a las siguientes personas: Max Burillo, William Vincent, Carlos Guardado, Enrique Cosio, Héctor Coronado, y Ana Isela Vargas, quienes me han bendecido con su amistad.

Ricardo Vargas de Basterra

Resumen

El desarrollo de software de calidad sigue siendo una constante preocupación en la industria de las Tecnologías de la Información y Comunicaciones. Mucho esfuerzo se ha realizado en esta materia con visiones tanto en el área de procesos constructivos como en la de producto terminado.

El desarrollo de software de calidad depende de múltiples factores que incluyen aspectos tanto teórico-metodológicos, instrumentales-tecnológicos y, por supuesto, de talento humano.

La habilidad para programar tiene sus orígenes en los procesos educativos que se desarrollan al interior de las Universidades que forman a los futuros ingenieros. Actualmente hay un preocupante decrecimiento en los índices de aprovechamiento ocasionando alta reprobación, abandono e incluso rechazo a la programación aún dentro de las carreras de esta profesión.

La programación es considerada una competencia en los perfiles profesionales e incluye, además de conocimientos, también habilidades, destrezas, valores y actitudes que combinadas le permiten a un ser humano producir código de calidad.

Esta tesis propone un modelo holístico que enfrenta el problema de la competencia de la programación en forma integral combinando tres grandes dimensiones:

- Teórica-metodológica.- Aportando una metodología denominada de microingeniería de software.
- Humana.- Aportando una taxonomía de habilidades para la programación y un método para su desarrollo y fortalecimiento.
- Herramental-tecnológica.- Aportando un lenguaje visual de modelado y una arquitectura para la construcción de un sistema de Diseño de Algoritmos Asistido por Computadora.

Todo esto apoyado en la propuesta de un conjunto de heurísticas que orienten sobre el diseño y modelado algorítmico imperativo.

El modelo se enfoca a la parte microscópica del área de ingeniería de software concentrándose específicamente en el análisis, diseño, construcción, pruebas y validación de los algoritmos inmersos en los métodos y funciones de las clases y otros componentes de software abordando la capa más específica y concreta de la producción de software.

Para su mejor comprensión, la tesis presenta un caso de estudio en el que se entrelazan, integralmente, los componentes del modelo para mostrar su forma de operar.

Finalmente, con la intención de verificar que los constructos teóricos del Modelo para la Construcción de Algoritmos Apoyado en Heurísticas son funcionales y prácticos se realiza un experimento de campo aplicado con rigor para validar, retroalimentar y perfeccionar el modelo. Este experimento, además de aportar la confiabilidad ya citada, aporta el diseño, prueba y aplicación de un instrumento para medir la habilidad humana para hacer diseño y construcción de algoritmos imperativos.

Abstract

The development of quality and reliable software is still a permanent concern in the Information and Communications Technology Industry. Many efforts have been taken in this matter, including visions in the construction processing, and as a final product.

The development of reliable software depends of multiple factors including issues in theory and methodology, technological tools and, of course, human talent.

The programming skill has its origins in the education process developed in the Universities where future engineers are being prepared. Nowadays, there is a concerning problem due to the decrease of the performance index causing a large number of students that fail, drop-out, and refusal to program in the computer science careers.

Programming is considered as a competence in the professional profiles including knowledge, skills, values, and attitudes; which combined allow a human being to produce quality and reliable software.

This thesis proposes a holistic model facing the programming competence problem combining, in an integral way, three large dimensions:

- Theory and methodology - Proposing a micro-software engineering methodology.
- Human – Proposing programming skills taxonomy and a development method.
- Technology and tools - Proposing a visual modeling language and architecture for building a Computer Aided Algorithms Design (DAAC for its Spanish initials) system.

All of the above supported on a proposed set of heuristics for orientation in modeling and design of imperative algorithms.

This model focuses on the microscopic area of software engineering. The analysis, design, development, tests and release of imperative algorithms in methods and functions in classes and software components are the main topic of this model. The most specific and concrete issue in software production is the microscopic point of view of this thesis.

For its better understanding, this thesis presents a case of study where all elements of the holistic model are combined in order to show how it works.

Finally, in order to verify that the theoretical elements of the thesis “Model for Building Algorithms Supported on Heuristics” are practical and functional, a field experiment has been performed. The methodological severity has been observed for validation, feedback, and improvement of the model. This experiment, besides contributing reliability on the model, proposes the design, test, and application of an instrument for measuring the human capability to build and develop imperative algorithms.

Tabla General de Contenido

Resumen.....	v
Abstract.....	vii
Índice Detallado	xi
Índice de Figuras.....	xvii
Índice de Tablas.....	xix
Glosario.....	xxi
Capítulo 1. Presentación del Problema.....	1
1.1 Antecedentes.....	1
1.2 Planteamiento del Problema.....	3
1.3 Hipótesis.....	12
1.4 Objetivos.....	13
1.5 Método.....	13
Capítulo 2. Marco Teórico: Reporte del Estado del Arte	19
2.1 Teoría de la Programación.....	19
2.2 Diseño de Algoritmos.....	33
2.3 Aprendizaje y Aplicación de la Programación.....	64
2.4 Herramientas Visuales de Programación.....	72
2.5 Reflexiones.....	77
Capítulo 3. Modelo para la Construcción de Algoritmos Apoyados en Heurísticas... ..	79
3.1 Introducción.....	79
3.2 Descripción del Modelo	80
3.3 Heurísticas	81
3.4 Dimensión Teórica: Microingeniería de Software	86
3.5 Dimensión Herramental: Diseño de Algoritmos Asistido por Computadora.....	136
3.6 Dimensión Humana: Modelo de Habilidades.....	151
Capítulo 4. Experimentación de Campo.....	177
4.1 Planeación del Experimento de Campo.....	177
4.2 Ejecución del Experimento de Campo	192
Capítulo 5. Conclusiones	207
5.1 Conclusiones del Modelo CAAH.....	207
5.2 Análisis de Resultados del Experimento de Campo.....	210
5.3 Alcance de objetivos.....	219
5.4 Aportaciones.....	222
5.5 Trabajos Futuros	226
5.6 Palabras Finales	228
Publicaciones	229
Revistas:	229
Informe Técnico:	229
Congresos:	229
Referencias.....	231
Anexos	237
A. Diseño de la Materia Principios de Programación.	239
B. Evaluación de Profesores.....	265
C. Prueba de Habilidades de Programación (HaDiCA)	279

Índice Detallado

Dedicatoria.....	i
Agradecimientos	iii
Resumen.....	v
Abstract.....	vii
Tabla General de Contenido	ix
Índice Detallado	xi
Índice de Figuras.....	xvii
Índice de Tablas.....	xix
Glosario.....	xxi
Capítulo 1. Presentación del Problema.....	1
1.1 Antecedentes.....	1
1.2 Planteamiento del Problema.....	3
1.2.1. Modelo Teórico	3
1.2.2. Modelo Humano.....	4
1.2.3. Modelo Tecnológico.....	5
1.2.4. Definición del Problema.....	6
1.2.5. Variables Independientes y Dependientes.....	7
1.2.6. Descripción del Trabajo	7
1.2.7. Aportaciones, Alcances y Limitaciones	9
1.2.7.1. Aportaciones y Beneficios.....	9
1.2.7.2. Alcances	9
1.2.7.3. Limitaciones	10
1.2.8. Justificación.....	10
1.2.8.1. Social	10
1.2.8.2. Profesional.....	11
1.2.8.3. Científica	12
1.3 Hipótesis	12
1.4 Objetivos.....	13
1.4.1. Principales	13
1.4.2. Específicos.....	13
1.5 Método.....	13
1.5.1. Población y Muestra	14
1.5.2. Definición de Variables.....	14
1.5.3. Diseño de Investigación	15
1.5.4. Control de Variables Extrañas.....	16
1.5.5. Instrumentos	16
1.5.6. Escenario	17
1.5.7. Procedimiento.....	17
Capítulo 2. Marco Teórico: Reporte del Estado del Arte	19
2.1 Teoría de la Programación.....	19
2.1.1. Conceptos	19
2.1.2. La Programación Como un Arte.....	20
2.1.3. Evolución de la Teoría de la Programación	20
2.1.4. Los Paradigmas de Programación	22

2.1.4.1. Paradigma Duro.....	22
2.1.4.2. Paradigma Libre	23
2.1.4.3. Paradigma Estructurado.....	23
2.1.4.4. Paradigma Orientado a Objetos.....	25
2.1.4.5. Otros Paradigmas.....	26
2.1.5. Evolución de los Lenguajes de Programación.....	27
2.1.5.1. De la Programación Dura al Macroensamblador	27
2.1.5.2. Del Macroensamblador a los Lenguajes de Alto Nivel. Primera Oleada.....	28
2.1.5.3. Segunda Oleada.....	29
2.1.5.4. Tercera Oleada.....	29
2.1.5.5. Los Lenguajes Descriptivos	30
2.1.5.6. Taxonomía de la Programación.....	30
2.1.6. El Linaje de la Programación	30
2.1.7. Mapa Mental de la Lógica de Programación.....	32
2.2 Diseño de Algoritmos.....	33
2.2.1. Conceptos	33
2.2.2. Métodos de Análisis de Problemas.....	34
2.2.3. Métodos de Diseño de Algoritmos	39
2.2.4. Herramientas y Técnicas	52
2.2.5. Heurísticas	61
2.2.6. Mapa Mental del Modelado Algorítmico	63
2.3 Aprendizaje y Aplicación de la Programación.....	64
2.3.1. Conceptos	64
2.3.2. Impacto en la Calidad del Software.....	64
2.3.3. Corrientes Educativas.....	65
2.3.3.1. Skinner.....	65
2.3.3.2. Piaget	65
2.3.3.3. Vigotsky	65
2.3.3.4. Bloom	66
2.3.3.5. Dubinsky.....	66
2.3.3.6. Bruner	67
2.3.3.7. Papert.....	67
2.3.3.8. Minsky.....	67
2.3.3.9. Anderson.....	68
2.3.3.10. Guilford	68
2.3.3.11. Gardner.....	69
2.3.4. Estrategias de Enseñanza Aprendizaje	70
2.3.5. Tácticas de Enseñanza Aprendizaje	70
2.3.6. Enfoques.....	71
2.4 Herramientas Visuales de Programación.....	72
2.4.1. MacGnome (1987).....	72
2.4.2. LabView (1990).....	73
2.4.3. Prograph (1992).....	73
2.4.4. Baci (1994).....	74
2.4.5. KidSim (1994).....	75
2.4.6. FlowCoder (1995)	76
2.4.7. ToonTalk (1995).....	77

2.4.8. Ambientes Visuales de Programación.....	77
2.5 Reflexiones.....	77
Capítulo 3. Modelo para la Construcción de Algoritmos Apoyados en Heurísticas...	79
3.1 Introducción.....	79
3.2 Descripción del Modelo	80
3.3 Heurísticas	81
3.3.1. Definiciones.....	83
3.3.2. Reglas Heurísticas de Modelado Algorítmico.....	85
3.4 Dimensión Teórica: Microingeniería de Software	86
3.4.1. Microingeniería	86
3.4.2. Análisis	88
3.4.2.1. Resultados.....	90
3.4.2.2. Procesos	90
3.4.2.3. Datos de Entrada.....	91
3.4.2.4. Restricciones.....	92
3.4.2.5. Repeticiones.	93
3.4.2.6. Diccionario de Datos	93
3.4.3. Diseño.....	94
3.4.3.1. Diseño de Secuencias	96
3.4.3.2. Diseño de Condicionales	97
3.4.3.3. Diseño de Ciclos.....	99
3.4.3.4. Diseño de Pruebas	99
3.4.3.5. Integración y Refinamiento	102
3.4.4. Construcción.....	103
3.4.5. Aplicación de Pruebas	105
3.4.6. Liberación.....	106
3.4.7. Caso de Estudio	106
3.4.7.1. Presentación del Problema: El Jardín.....	106
3.4.7.2. 1ª Iteración.....	107
3.4.7.2.1. Fase de Análisis.....	107
3.4.7.2.1.1. Identificación de Salidas.....	107
3.4.7.2.1.2. Identificación de Procesos	107
3.4.7.2.1.3. Identificación de Entradas	108
3.4.7.2.1.4. Diccionario de Datos	109
3.4.7.2.2. Fase de Diseño.....	110
3.4.7.2.2.1. Diseño de Secuencias	110
3.4.7.2.2.2. Diseño de Pruebas	112
3.4.7.2.3. Fase de Construcción.....	113
3.4.7.2.4. Fase de Pruebas	114
3.4.7.2.5. Fase de Liberación.....	114
3.4.7.3. 2ª Iteración.....	115
3.4.7.3.1. Fase de Análisis.....	115
3.4.7.3.1.1. Identificación de Restricciones.....	115
3.4.7.3.2. Fase de Diseño.....	118
3.4.7.3.2.1. Diseño de Restricciones	118
3.4.7.3.2.2. Diseño de Pruebas	119
3.4.7.3.3. Fase de Construcción.....	120

3.4.7.3.4. Fase de Pruebas	122
3.4.7.3.5. Fase de Liberación.....	122
3.4.7.4. 3ª Iteración.....	123
3.4.7.4.1. Fase de Análisis.....	123
3.4.7.3.4.1. Identificación de Repeticiones	123
3.4.7.4.2. Fase de Diseño.....	124
3.4.7.4.2.1. Diseño de Repeticiones	124
3.4.7.4.2.2. Diseño de Pruebas	127
3.4.7.4.3. Fase de Construcción.....	127
3.4.7.4.4. Fase de Pruebas	129
3.4.7.4.5. Fase de Liberación.....	130
3.4.7.5. Producto Final	131
3.4.7.5.1. Fase de Análisis.....	131
3.4.7.5.2. Fase de Diseño.....	133
3.4.7.5.3. Fase de Construcción.....	134
3.4.7.5.4. Fase de Pruebas	136
3.4.7.5.5. Fase de Liberación.....	136
3.5 Dimensión Herramental: Diseño de Algoritmos Asistido por Computadora.....	136
3.5.1. La Programación Visual Imperativa.....	136
3.5.1.1. Definición de Lenguaje de Programación Visual.....	137
3.5.1.2. Expectativas sobre la Programación Visual	138
3.5.1.3. Desventajas y Problemática.....	139
3.5.1.4. Alternativas de Solución Propuestas	141
3.5.2. Lenguaje Visual de Modelado.....	142
3.5.2.1. Gramática Visual	144
3.5.2.2. Gramática de Enunciados	147
3.5.2.3. Semántica	148
3.5.3. Arquitectura del Sistema	150
3.5.3.1. Arquitectura DAAC.....	150
3.6 Dimensión Humana: Modelo de Habilidades.....	151
3.6.1. La Lógica de Programación como una Habilidad Mental.....	152
3.6.2. Un Modelo para Clasificar las Habilidades.....	152
3.6.2.1. Contenidos de la Información	154
3.6.2.2. Productos de la Información.....	154
3.6.2.3. Procesos Intelectuales.....	155
3.6.2.4. Habilidades Intelectuales en la Programación.....	156
3.6.3. Modelo de Desarrollo de Habilidades	159
3.6.3.1. Dos modelos de Educación	160
3.6.3.2. Construyendo el Conocimiento	162
3.6.4. Taxonomía de Habilidades	164
3.6.5. Desarrollando las Habilidades para la Programación.....	173
3.6.5.1 Estilos de Aprendizaje.....	173
3.6.5.2. Modelos de Estilos de Aprendizaje	173
3.6.5.3. Estrategia de Desarrollo de Habilidades de Programación	175
Capítulo 4. Experimentación de Campo.....	177
4.1 Planeación del Experimento de Campo.....	177
4.1.1. Planeación del Control de las Variables Extrañas.....	178

4.1.1.1. Control de Antecedentes.....	178
4.1.1.2. Control de Maduración.....	179
4.1.1.3. Control de Bajas	179
4.1.1.4. Control de Instrumentación	179
4.1.1.5. Control de Reactividad al Estimulo.....	179
4.1.2. Diseño y Planeación Didáctica del Curso	180
4.1.3. Integración de Grupos de Control y Experimental.....	180
4.1.4. Selección de Profesores.....	181
4.1.4.1. Definición de Criterios	181
4.1.4.2. Aplicación de los Criterios	181
4.1.4.3. Selección Propuesta.....	183
4.1.5. Selección y Aplicación de Instrumentos de Preprueba y Posprueba.....	184
4.1.5.1. Batería de Pruebas para Determinar el Perfil de Ingreso (Yb).....	184
4.1.5.2. Batería de Pruebas para Determinar el Perfil de Egreso (Ya).....	184
4.1.5.3. Condiciones de Aplicación.....	185
4.1.5.4. Prueba de Habilidades de Diseño y Construcción de Algoritmos (HaDiCA).185	
4.1.5.4.1. Especificaciones de HaDiCA	185
4.1.5.4.2. Diseño de HaDiCA.....	186
4.1.5.4.3. Aplicación, Calificación e Interpretación de HaDiCA.....	188
4.1.6. Plan de Introducción del Modelo de Construcción de Algoritmos Apoyados en Heurísticas.....	191
4.1.7. Calendario.....	191
4.2 Ejecución del Experimento de Campo	192
4.2.1. Acciones de Control de Variables Extrañas	192
4.2.1.1. Integración de Grupos	192
4.2.1.2. Control de Antecedentes.....	193
4.2.1.2.1. Número Total de Estudiantes	194
4.2.1.2.2. Distribución de Edades.....	194
4.2.1.2.3. Resultados de la Prueba de Pronóstico de Éxito Académico	194
4.2.1.2.4. Resultado de la Prueba de College Board	195
4.2.1.2.5. Resultado de la Prueba de Antecedentes en Cultura Informática.....	197
4.2.1.3. Control de Maduración.....	200
4.2.1.4. Control de Bajas	201
4.2.1.5. Control de Instrumentación	202
4.2.1.6. Control de Reactividad al Estímulo.....	202
4.2.2. Acciones de la Planeación Didáctica.....	203
4.2.3. Conclusiones de la Ejecución del Experimento	204
Capítulo 5. Conclusiones	207
5.1 Conclusiones del Modelo CAAH.....	207
5.2 Análisis de Resultados del Experimento de Campo.....	210
5.2.1. Análisis de Resultados de Yb (al inicio del experimento)	210
5.2.1.1. Índices Iniciales de PEA.....	210
5.2.1.2. Índices Iniciales de College Board.....	210
5.2.1.3. Índices Iniciales de Prueba de Antecedentes Informáticos	211
5.2.2. Análisis de Resultados de Ya (prueba de habilidades al final de proceso)	212
5.2.2.1. Resultados del Grupo Experimental (02C).....	212
5.2.2.2. Resultados del Grupo de Control (04c).....	214

5.2.2.3. Análisis Comparativo	215
5.2.3. Prueba de Hipótesis	218
5.3 Alcance de objetivos.....	219
5.3.1. De los Objetivos Generales.	219
5.3.2. De los Objetivos Específicos.....	220
5.4 Aportaciones.....	222
5.4.1. Dentro del Marco Teórico:	222
5.4.2. Dentro del Marco del Talento Humano:.....	223
5.4.3. Dentro del Marco Herramental.....	223
5.4.4. Dentro del Contexto Experimental.....	224
5.4.5. Otros Beneficios y Aportaciones.....	225
5.5 Trabajos Futuros.....	226
5.6 Palabras Finales	228
Publicaciones	229
Revistas:	229
Informe Técnico:	229
Congresos:	229
Referencias.....	231
Anexos	237
A-1 Planeación didáctica del grupo de control.....	239
A-2 Planeación didáctica del grupo experimental.....	249
A-3 Paquete de prácticas de laboratorio.....	259
B-1 Formato de evaluación de profesores	265
B-2 Resultados de le evaluación previa al experimento (Ciclo 02-05).....	271
B-3 Resultados de la evaluación posterior al experimento (Ciclo 03-05).....	275
C-1 Hoja de problemas.....	279
C-2 Diseño de rúbricas de evaluación.....	283
C-3 Solución de problemas	291
C-4 Formato para concentrar datos de la aplicación.....	323

Índice de Figuras

Figura 1. Trípode de la programación.....	2
Figura 2. Estructuras de control para el paradigma estructurado	24
Figura 3. Taxonomía de los modelos de programación con ejemplos	31
Figura 4. Lógica de programación.....	32
Figura 5. Linaje de los lenguajes de programación.....	33
Figura 6. Sintaxis de los diagramas de Warnier/Orr	41
Figura 7. Diagrama de Warnier/Orr. Ejemplo de programa para ordenar números.....	42
Figura 8. Símbolos de Jackson	43
Figura 9. Estructuras de control de Jackson	44
Figura 10. Diagrama de Jackson	45
Figura 11. Estructuras de control de Bertini.....	46
Figura 12. Diagrama de Bertini	47
Figura 13. Estructuras de control de Tabourier	48
Figura 14. Diagrama de Tabourier	49
Figura 15. Símbolos utilizados para los diagramas de flujo propuestos por IBM	52
Figura 16. Símbolos de diagramas de flujo ANSI.....	53
Figura 17. Estructuras de control del paradigma libre	53
Figura 18. Ejemplo de diagrama de flujo en el paradigma libre.	54
Figura 19. Ejemplo de grafo.....	55
Figura 20. Tabla de decisión	55
Figura 21. Árboles de decisiones.....	56
Figura 22. Estructuras de control en ordinogramas.....	57
Figura 23. Ordinograma	58
Figura 24. Estructuras de control de Chapin (Nassi/Shneiderman)	59
Figura 25. Diagrama de Chapin (Nassi/Shneiderman).....	60
Figura 26. Diagrama de actividad de UML.....	61
Figura 27. Modelado algorítmico.....	63
Figura 28. Ejemplo de MacGnome	72
Figura 29. Ejemplo de LabView	73
Figura 30. Ejemplo de Prograph.....	74
Figura 31. Ejemplo de BACII	75
Figura 32. Ejemplo de KidSim.....	76
Figura 33. Ejemplo de FlowCoder	77
Figura 34. Modelo para la construcción de algoritmos apoyado en heurísticas.....	81
Figura 35. Ejemplos de símbolos que representan operaciones	83
Figura 36. Estructuras de control.....	84
Figura 37. Diagrama de tarea	84
Figura 38. Símbolos de acciones.	85
Figura 39. Metodología de microingeniería de software.....	87
Figura 40. Modelo del análisis	88
Figura 41. Árbol de descomposición de procesos.....	90
Figura 42. Diseño conceptual del viaje de la computadora.....	94
Figura 43. Modelo de diseño de algoritmos	95
Figura 44. Modelo de tránsito del análisis al diseño	95

Figura 45. Ejemplos de diagramas de secuencias	97
Figura 46. Diseños posibles de condicionales	98
Figura 47. Diseños de ciclos.....	100
Figura 48. Diseño de bifurcaciones anidadas	103
Figura 49. Árbol de descomposición de procesos para presupuesto	108
Figura 50. Árbol de descomposición de procesos para tmc	108
Figura 51. Primeros diagramas de secuencias	110
Figura 52. 1ª versión de El Jardín.....	112
Figura 53. Caso de prueba 1. Validación de secuencia	115
Figura 54. Diseño de procedimiento condicional.....	118
Figura 55. Integración y refinamiento para 2a iteración	119
Figura 56. Caso de prueba 2 de 2a iteración	122
Figura 57. Caso de prueba 3 de 2a iteración	122
Figura 58. Integración y refinamiento 3a iteración	125
Figura 59. Diagrama final del problema El Jardín	126
Figura 60. Caso de prueba cuatro de El Jardín.....	130
Figura 61. Caso de prueba cinco de El Jardín	130
Figura 62. Caso de prueba seis de El Jardín.....	131
Figura 63. Versión 1.0 final de El Jardín.....	133
Figura 64. Gramática de LeMVI	145
Figura 65. Elemento documental de LeMVI.....	145
Figura 66. Reglas de producción de la gramática de LeMVI (conjunto P).....	146
Figura 67. Arquitectura conceptual	150
Figura 68. Arquitectura DAAC	151
Figura 69. Estructura de la inteligencia de Guilford	153
Figura 70. Flujo de procesos mentales	156
Figura 71. Habilidades intelectuales en el análisis.....	169
Figura 72. Habilidades intelectuales en el diseño	170
Figura 73. Habilidades intelectuales en la etapa de pruebas	170
Figura 74. Habilidades en la programación.....	171
Figura 75. Proporción de habilidades por fase	172
Figura 76. Modelo incremental de habilidades por fase.....	175
Figura 77. Perfil por habilidades	190
Figura 78. Perfil por descriptores	191
Figura 79. Perfiles de sistemas operativos	197
Figura 80. Perfiles en ofimática.....	198
Figura 81. Perfiles en Internet	199
Figura 82. Perfiles en programación	199
Figura 83. Histograma de habilidades específicas grupo experimental	213
Figura 84. Diagrama de frecuencias del grupo experimental.....	213
Figura 85. Histograma de habilidades específicas grupo de control	214
Figura 86. Diagrama de frecuencias de grupo de control.....	215
Figura 87. Diagrama de frecuencias comparativo.....	217
Figura 88. Comparativo de índices.....	218
Figura 89. Perfil comparativo.....	218

Índice de Tablas

Tabla 1. HIPO módulo principal	50
Tabla 2. HIPO Proceso de ordenación	51
Tabla 3. HIPO Proceso alternativo B	51
Tabla 4. HIPO Proceso alternativo C	51
Tabla 5. Características de los sistemas visuales.....	78
Tabla 6. Tabla de resultados.....	90
Tabla 7. Tabla de entradas.....	91
Tabla 8. Diccionario de datos.....	93
Tabla 9. Condición simple y condición compuesta.....	98
Tabla 10. Ejemplo de transcripción a Pascal.....	104
Tabla 11. Ejemplo de transcripción a C#	104
Tabla 12. Tabla de salidas.	107
Tabla 13. Tabla en entradas.....	109
Tabla 14. Diccionario de datos 1a 1ª versión	109
Tabla 15. Diccionario de datos 2a versión	116
Tabla 16. Diccionario de datos versión final.....	132
Tabla 17. Principios de Fitter	139
Tabla 18. Soluciones a problemas de modelado visual de algoritmos	142
Tabla 19. Especificaciones comparativas del lenguaje	143
Tabla 20. Operadores y operandos	148
Tabla 21. Habilidades intelectuales de captación de información	157
Tabla 22. Habilidades intelectuales de la memoria.....	157
Tabla 23. Habilidades intelectuales de la evaluación de información.....	158
Tabla 24. Habilidades de producción convergente para la programación.....	158
Tabla 25. Habilidades de producción divergente para la programación	159
Tabla 26. Habilidades intelectuales que corresponden a las técnicas	167
Tabla 27. Habilidades intelectuales requeridas por fase	169
Tabla 28. Porcentaje de errores en la programación	172
Tabla 29. Estilos de aprendizaje.....	175
Tabla 30. Integración de grupos	181
Tabla 31. Cuadro resumen del perfil de profesores candidatos.....	183
Tabla 32. Puntaje de HaDiCA	188
Tabla 33. Ejemplo de puntaje.....	190
Tabla 34. Calendario de actividades.....	192
Tabla 35. Grupos integrados con profesor	193
Tabla 36. Población en los grupos candidatos.....	194
Tabla 37. Análisis estadístico de edades en la población.....	194
Tabla 38. Resultados de evaluación PEA.....	195
Tabla 39. Análisis estadístico de la prueba de aptitudes verbales.....	195
Tabla 40. Resultados de prueba <i>t</i> de <i>student</i> de aptitudes verbales.....	196
Tabla 41. Análisis estadístico de la prueba de habilidades numéricas	196
Tabla 42. Resultaos de prueba <i>t</i> de <i>student</i> de aptitudes numéricas.....	197
Tabla 43. Evaluación de antecedentes técnicos.....	200
Tabla 44. Análisis de promedios finales.....	200

Tabla 45. Resultados en prueba de antecedentes aplicada al inicio del ciclo.....	201
Tabla 46. Análisis comparativo de maduración	201
Tabla 47. Control de bajas.....	201
Tabla 48. Resumen de control de variables extrañas	206
Tabla 49. Integración del modelo CAAH.....	209
Tabla 50. Índices de la prueba PEA	210
Tabla 51. Índices de la prueba College Board.....	211
Tabla 52. Promedio de evaluación de antecedentes técnicos	211
Tabla 53. Índice HaDiCA grupo experimental.....	212
Tabla 54. Índice HaDiCA grupo de control	214
Tabla 55. Comparativo de índice HaDiCA	216
Tabla 56. Análisis estadístico comparativo	216
Tabla 57. Prueba <i>t</i> de <i>student</i> de índices HaDiCA entre grupos Ge y Gc.....	219

Glosario

Acción .- Es un subconjunto de las operaciones en las que solamente se consideran las de entrada, de proceso y de salida.

Algoritmo.- Conjunto ordenado de instrucciones que tiene las siguientes características: es preciso, genera resultados únicos, es finito, tiene un cierto número de entradas, produce al menos una salida o resultados, es generalizable.

Análisis.- Identificación de las partes que conforman un todo. Es la primera fase de la metodología de microingeniería.

Árbol de Descomposición de Procesos.- Diagrama tipo árbol que describe los cálculos para llegar a un resultado a partir de los datos de entrada.

College Board.- Prueba internacional estandarizada que mide las habilidades verbales y las habilidades numéricas de estudiantes aspirantes a ingresar a una carrera universitaria.

Competencia.- Capacidad productiva de un individuo que se define y mide en términos de desempeño en un determinado contexto laboral. Incluye un conjunto de habilidades, destrezas, actitudes valores y conocimientos.

Construcción.- Es la tercera fase de la metodología de microingeniería. Describe el proceso de convertir el diseño de un algoritmo de lenguaje simbólico a código en algún lenguaje que la computadora pueda compilar y ejecutar.

Construcción de Algoritmos Apoyado en Heurísticas CAAH.- Modelo holístico integrado por componentes teóricos, humanos, herramientas y heurísticas para poder analizar problemas, diseñar algoritmos, codificarlos en algún lenguaje imperativo, probarlo y liberar el producto final.

Contenidos de información.- Uno de los tres ejes del modelo de la inteligencia de Guilford. Toda información tiene necesariamente un contenido específico que diferenciará la estructura intrínseca de los datos que posteriormente procesará la inteligencia. Hay cuatro tipos de contenidos: Información figurativa, Información simbólica, Información semántica e Información Conductual.

Descriptor.- La prueba de Habilidades de Diseño y Construcción de Algoritmo (HaDiCA) mide tres grandes habilidades: Análisis, Diseño y Construcción. Cada una de estas se subdivide en descriptor que son componentes específicos de cada habilidad. Así, los descriptor de HaDiCA para el análisis son: resultados, procesos, entradas necesarias, restricciones y/o condiciones, iteraciones o repeticiones y diccionario de datos; los descriptor para el diseño son: secuencias, condiciones, iteraciones, pruebas, integración y consistencia; los descriptor para la construcción son: funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y consistencia.

Diccionario de Datos.- Es una tabla en la que se relacionan todos los datos que se utilizan en un algoritmo. Para cada dato se agregan descripciones, tipos, dominios, uso, etc.

Dimensión.- En los modelos holísticos es el nombre que se le da a cada uno de los grandes componentes que forman al modelo. En el Modelo CAAH son tres grandes dimensiones las que lo integran: modelo teórico, modelo humano y modelo herramental. Todo esto apoyado en un conjunto de heurísticas.

Diseño.- Es la segunda fase de la metodología de microingeniería. Consiste en modelar y definir cada una de las partes que conforman al algoritmo.

Diseño de Algoritmos Asistido por Computadora DAAC.- Arquitectura propuesta para construir un sistema de software que facilite el análisis, diseño, construcción, pruebas y liberación de algoritmos imperativos. Incluye un conjunto de heurísticas y un lenguaje de modelado visual.

Dominio.- Conjunto de valores válidos (rango) que una variable puede asumir (típicamente la variable independiente de una función). También se le considera el grado de aprendizaje que tiene un ingeniero de software sobre las herramientas, técnicas, métodos, etc.

Entrada.- Operación que representa la lectura de uno o más valores desde algún dispositivo para ser almacenado en una o más variables.

Estilos de Aprendizaje.- Rasgos cognitivos, afectivos y fisiológicos que sirven como indicadores relativamente estables, de cómo los alumnos perciben interacciones y responden a sus ambientes de aprendizaje. Hay varias propuestas de diferentes estilos de aprendizaje, una de las más comunes, el de la programación neurolingüística, propone tres estilos: auditivo, visual y kinestésico.

Estructura de Control.- Conjunto de operaciones que permiten representar alguna de las tres estructuras indicadas en el teorema de Bohēm-Jacopini: secuencia, condición y repetición. Actualmente se identifican tres tipos de condición y tres tipos de repetición haciendo un total de siete estructuras de control: secuencia, if-then, if-then-else, case, for, while y until. Las estructuras de control se consideran unidades de programación indivisibles.

Gramática.- Descripción formal de los componentes de un lenguaje los cuales están formados por elementos léxicos, sintácticos, semánticos y reglas de producción. LeMVI utiliza una gramática libre de contexto que se define $G = (N, T, S, P)$, donde N es el conjunto de símbolos no terminales; T es un conjunto de componentes léxicos denominados símbolos terminales.; S es un símbolo no terminal utilizado como inicial y P es un conjunto de reglas de producción.

Habilidad.- Viene del término latino *habilis* que significa manejable. Se define como la capacidad, disposición, gracia o destreza para hacer algo; en este caso programar.

Habilidades de la programación.- Del universo de habilidades mentales descritas en el modelo de Guilford (más de 100), 49 se han identificado que se aplican en el proceso de programación y de ellas nueve son las de alto impacto.

Habilidades de Diseño y Construcción de Algoritmos (HaDiCA).- Es una prueba que mide las habilidades para analizar, diseñar y construir un algoritmo a partir de un problema. Cada habilidad se subdivide en seis descriptores (ver definición de descriptores del glosario) y cada uno tiene una escala de likert de 1 a 4. La prueba aplica al menos dos escenarios teniendo un puntaje mínimo de 36 puntos y un máximo de 144.

Heurística.- Heurística es una regla sencilla y eficiente que pueda orientar en el proceso de diseño de un algoritmo para la resolución de problemas computables, en los que los diseños se descubren por la evaluación del progreso logrado en la búsqueda de un algoritmo final

Holístico.- Proviene del griego *holos* (όλος) que significa todo. Es el estudio del todo, relacionándolo con sus partes pero sin separarlo del todo.

Índice HaDiCA.- Puntaje acumulado que obtiene un grupo de personas al aplicar la prueba HaDiCA. La escala va de 36 a 144 puntos medidos en 18 descriptores y 3 habilidades utilizando Rúbricas y escalas de likert.

Iteraciones.- La metodología de microingeniería utiliza un proceso repetitivo e incremental que recibe el nombre de iteración. En cada iteración se repiten las fases de análisis, diseño, construcción y pruebas. Para obtener un algoritmo terminado y depurado se requieren varias repeticiones del proceso y cada una recibe el nombre de iteración.

Lenguaje de Programación Visual (LPV).- Conjunto de diagramas u objetos gráficos con los que pueden componerse sentencias válidas en un lenguaje de programación. En inglés se les denomina VPL's por sus siglas.

Lenguaje de Modelado Visual Imperativo (LeMVI).- Es un lenguaje visual, inspirado en los tradicionales diagramas de flujo de control y por ello se le define como imperativo. Presenta características visuales tales que no permite la ambigüedad simbólica ni la libertad estructural diagramática aportando mejoras de interpretación y estructura. Está basado en el teorema de Jacopini.

Liberación.- Quinta fase de la metodología de microingeniería. En ella se integra toda la documentación creada en cada fase previa y se libera el producto controlando los números de versión y las modificaciones realizadas.

Lógica de programación.- Habilidad mental utilizada para implementar soluciones computacionales (programas) orientados a ejecutar tareas complejas funcionalmente verdaderas (que hagan lo que se espera de ellos - Conclusiones válidas) a través de la combinación de diferentes estructuras de control y estructuras de datos (premisas) combinadas utilizando una serie de reglas formales (heurísticas o "Reglas de la Lógica").

Método.- Procedimiento sistematizado que se utiliza para alcanzar un determinado fin.

Metodología.- Conjunto de métodos, procedimientos y herramientas que se utilizan en una investigación o proceso.

Microingeniería.- La metodología de microingeniería de software aspira a trazar una ruta formal que conduzca de forma reflexiva a una persona del dominio del problema al dominio de la solución. Recibe el nombre de microingeniería porque sus fronteras inician en las definiciones de los contratos que los métodos de las clases deben cumplir (incluyendo pre y post condiciones, así como el respeto del estado de los objetos) y terminan en la implementación de dicho método. Atiende la parte más concreta del proceso total de ingeniería.

Modelado Algorítmico.- Proceso de diseño de algoritmos utilizando herramientas específicas para ello tal como un lenguaje visual.

Modelo.- Reproducción ideal y concreta de un objeto o de un fenómeno con fines de estudio y experimentación. La palabra modelo viene del latín *modulus* que significa molde. Dependiendo de su naturaleza estará formado por un conjunto de componentes y reglas que rigen su interacción.

Operación.- Actividad específica que realiza la computadora por indicaciones del programador. En un lenguaje visual cada operación se representa por un símbolo específico.

Paradigma.- Paradigma proviene del vocablo griego *paradeigma* (*παράδειγμα*) que significa mostrar o manifestar. Se ha utilizado como conjunto de reglas y principios que rigen una forma de pensar y de actuar.

Procedimiento.- Conjunto de estructuras de control organizadas y secuenciadas de tal forma que muestran la solución parcial de una tarea.

Proceso.- Operación que representa la ejecución de un cálculo y asignación del resultado a una variable. También puede representar la invocación de un método de un objeto o de una función.

Productos de la información.- Los productos se refieren a la estructura de la información: son formas de construcción mental. Hay seis diferentes tipos de productos: Unidades, Clases, Relaciones, Sistemas, Transformaciones e Implicaciones.

Programación.- Proceso mental complejo, dividido en las siguientes etapas: análisis, diseño, codificación, pruebas y liberación. La finalidad de la programación es comprender con claridad el problema que va a resolverse o simularse por medio de la computadora, y entender también con detalle cuál será el procedimiento mediante el cual la máquina llegará a la solución deseada.

Pronostico de Éxito Académico (PEA).- Prueba estandarizada que se aplica a estudiantes aspirantes a ingresar a nivel universitario que estima las posibilidades de éxito o fracaso medidas en términos de aptitudes hacia el estudio.

Pruebas.- Son mecanismos de validación del algoritmo. El objetivo es el de detectar todo posible malfuncionamiento antes de considerar un algoritmo terminado. Esto aportará validez y confianza al algoritmo que se está diseñando. Probar un programa es ejercitarlo con la peor intención a fin de encontrarle fallos.

Reglas de la lógica.- Conjunto de doce heurísticas que orientan sobre el modelado y diseño de un algoritmo.

Repeticiones.- Conjunto de instrucciones que deben ejecutarse en varias ocasiones dentro de un algoritmo. Las estructuras de repetición se dividen en dos categorías: cerradas y abiertas. Las de repeticiones cerradas son aquellas que señalan de antemano el número de repeticiones que deben hacerse; Las de repeticiones abiertas son aquellas en las que se desconoce el número de iteraciones.

Restricciones.- Las restricciones son condiciones que los datos deben de satisfacer para poder ser considerados como válidos para el proceso. Estas restricciones pueden pertenecer al dominio del problema (mundo real) o al dominio de la solución (mundo virtual). Las restricciones se escriben en forma de condición aplicando el dominio al que pertenece.

Rúbrica.- Es una matriz de valoración que facilita la calificación del desempeño del estudiante en las áreas del currículo (materias o temas) que son complejas, imprecisas y subjetivas. Esta Matriz podría explicarse como un listado del conjunto de criterios específicos y fundamentales que permiten valorar el aprendizaje, los conocimientos y/o las competencias, logrados por el estudiante en un trabajo o materia particular. La rúbrica está compuesta por un conjunto de criterios y una escala de medición con descriptores claramente definidos (ver definición de descriptores del glosario), en este caso corresponde a las habilidades que se medirán en la prueba.

Salida.- Operación que representa la visualización del valor de uno o más datos en algún dispositivo.

Secuencias.- Estructura de control que contiene un conjunto de instrucciones que se ejecutan una inmediatamente después de la otra y típicamente incluyen la entrada, el proceso y al salida de ciertos resultados. Cada resultado tiene su propia secuencia de cálculo por lo que se podrían crear varios diseños de secuencias, al menos uno precisamente para cada resultado.

Tarea.- Conjunto de estructuras de control cuyo inicio y fin están claramente indicadas por elementos terminales. Típicamente representan la solución completa a un problema o acción que debe resolverse o ejecutarse por un equipo de cómputo.

Taxonomía de Habilidades.- Medio de clasificación de las habilidades necesarias para la programación definidas en el contexto de una metodología de diseño algorítmico. La competencia de la programación se desglosa en una serie de habilidades técnicas, las cuales a su vez se componen de habilidades intelectuales.

Teoría de la programación.- conjunto de principios que rigen un paradigma de programación y definen las características semánticas de cierto conjunto de lenguajes. La teoría de la programación aspira a dar certidumbre sobre las formas más adecuadas de enfrentar un problema, modelarlo y encontrar su solución algorítmica para poder codificarlo en algún lenguaje y ejecutar la solución en una computadora.

Variable Extraña.- Factores externos al estímulo que pueden afectar al resultado y ocasionar la pérdida de validez del experimento. Para disminuir su influencia en la variable dependiente se consideraron varias estrategias de control.

Modelo para la Construcción de Algoritmos Apoyado en Heurísticas.

Capítulo 1. Presentación del Problema

1.1 Antecedentes

Todos los analistas concuerdan que el componente más costoso de un sistema de información es el software. Esto ha llevado a muchos investigadores y profesionales del área a buscar metodologías, técnicas, herramientas, leyes, principios y todo un conjunto de elementos que permitan la construcción de software eficiente, barato, mantenible, libre de errores y eficaz, en pocas palabras software de calidad.

El problema de la calidad del software ha tomado dos grandes vertientes: una de ellas ve al software como un producto terminado que debe cumplir con una serie de especificaciones claras prácticamente dictadas desde su concepción, por ejemplo la norma ISO 9126 señala seis categorías de criterios de calidad que el producto terminado debe poseer; La otra vertiente es garantizar la calidad del software vigilando su proceso de fabricación poniendo atención a las normas, procedimientos, metodologías y habilidades necesarias, por ejemplo el CMM (en español Modelo de Madurez de Capacidades) señala cinco categorías en que es posible clasificar a un área de desarrollo de software en función del grado de madurez que tiene en el manejo de sus procesos de manufactura. A pesar de la divergencia y, gracias a ella, complementariedad que tienen ambas visiones, existe un elemento común que opera como variable clave y determinante en ellas: dependen del talento humano.

La ingeniería de software ha hecho grandes aportaciones para el avance del desarrollo de sistemas. A lo largo de los años y desde las primeras teorías de programación hasta la actualidad, ha habido una constante evolución partiendo de la forma para modelar el dominio del problema hasta las técnicas específicas de programación bajo algún paradigma.

El diseño de algoritmos es una parte crítica en el proceso de ingeniería de software. Con el tiempo, han surgido diversos enfoques, técnicas y teorías formando un paraguas para la concepción de herramientas de diseño y construcción. Muchas de estas metodologías nacieron dentro del seno de alguna teoría de programación misma que a su vez gestó algún paradigma. El hecho de estar asociado a un paradigma particular limitaba su vigencia a las fronteras del propio paradigma lo que ocasionó el nacimiento y muerte, en ocasiones en forma prematura, de algunas ideas y principios. Esta situación ha limitado la capacidad que se tiene de automatizar el proceso de diseño algorítmico.

La consecuencia del diseño de algoritmos y su escritura en una notación que permita su ejecución en computadora se llama programación. La programación sigue siendo el corazón de todo proyecto de software. En la sección del Estado del Arte se hace una recopilación histórica, desde 1945 a la fecha, de las principales teorías de programación y como estas han dictado la creación de nuevos lenguajes, paradigmas, técnicas, herramientas y las principales metodologías

de diseño algorítmico que eventualmente se convierten en los tópicos curriculares de los centros educativos.

Ha pesar de los grandes avances en la forma de analizar y diseñar sistemas de computadora se encontró, durante la investigación del estado del arte, que las técnicas y herramientas para el diseño algorítmico han tenido sus claroscuros con momentos de gran auge hasta otros de casi total abandono.

La capacidad de los seres humanos para entender un problema, modelarlo, diseñar una solución e implementarla en forma de un programa de computadora, en una palabra programar, se ve determinada por diversos factores que se resumen en el modelo de la figura 1 denominado el Trípode de la Programación. Este modelo tiene tres vértices formado por los siguientes componentes:

- A. Marco Teórico, formado por las teorías, paradigmas, procedimientos y metodologías que conforman un marco conceptual sobre el cual se modela el dominio del problema y el dominio de la solución.
- B. Talento Humano. Que representa las habilidades individuales que se poseen, y en particular las habilidades de programación;
- C. Herramientas, formadas por distintos componentes de hardware y software que permiten construir, implementar y operar la solución.

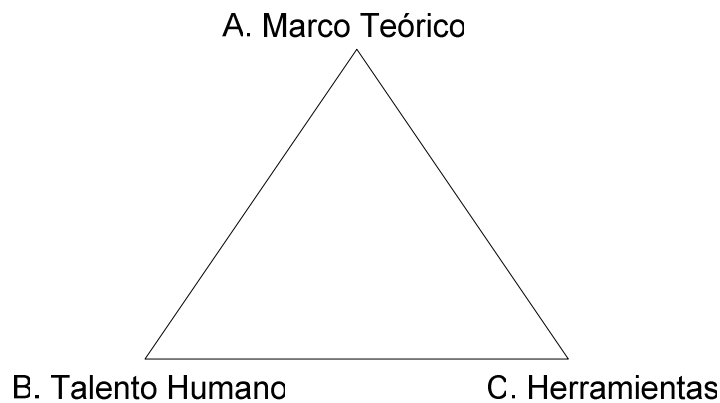


Figura 1. Trípode de la programación.

La correcta combinación entre estos tres elementos afectan directamente las probabilidades de éxito de un proyecto de Ingeniería de Software. En la medida que se cuente con un modelo integral se estará en condiciones de incidir significativamente en la calidad del software.

La programación, en consecuencia, es el producto final de los procesos de ingeniería de software, incluyendo el diseño algorítmico. Hoy día, la capacidad de automatizar la elaboración de código completo¹ depende de la habilidad para diseñar algoritmos y codificarlos en forma

¹ Muchos sistemas de tipo CASE aseguran poder realizar ingeniería hacia adelante creando código fuente en lenguaje 3GL a partir de los modelos creados con alguna metodología. Sin embargo, su capacidad está limitada a la

asistida por computadora. Esto implica la necesidad de abordar cada uno de los vértices del modelo anterior planteando lo siguiente: para el marco teórico, una serie de formalismos, un conjunto de heurísticas y una metodología de micro ingeniería de software; para el talento humano, una taxonomía de habilidades para la programación y una metodología de desarrollo de habilidades mentales superiores; y para las herramientas un lenguaje visual de modelado y una arquitectura que permita la construcción de herramientas de software de Diseño de Algoritmos Asistido por Computadora². Estos componentes forman el Modelo para la Construcción de Algoritmos Apoyado en Heurísticas.

1.2 Planteamiento del Problema

Siendo el software un intangible que requiere para su construcción teorías, conocimientos, herramientas y mucho talento y habilidades, ¿Hay alguna forma de mejorar el proceso para diseñar y construir algoritmos imperativos? La respuesta a esta pregunta debe ser sí.

Esta investigación busca definir un modelo para la Construcción de Algoritmos Apoyado en Heurísticas y demostrar que mejora la habilidad humana para diseñar algoritmos. Este modelo contiene una serie de elementos que combinados abordan de forma integral los tres vértices del Trípode de la Programación citados en la figura 1. Cada uno de estos vértices presenta sus propios retos lo que permite enfocar el planteamiento del problema en tres partes:

- Modelo Teórico
- Modelo Humano
- Modelo Tecnológico

1.2.1. Modelo Teórico

El objeto de esta tesis es presentar un modelo para la construcción de algoritmos integral que a través de un conjunto de heurísticas aplicadas mejore la habilidad humana para programar. De aquí lo primero que hay que abordar es que se entenderá por heurística.

La palabra heurística proviene del griego *heurísto* (*εὐρίστω*) y significa inventar. Este término ha sido utilizado en muchas formas gramaticales y por distintas ciencias. En el caso de estudio que se aborda se debe considerar el enfoque computacional acompañado del enfoque psicológico ya que el objetivo es la identificación de heurísticas que dirijan el diseño de algoritmos pero desde la perspectiva humana.

sección de la interfaz de la clase que se modela. La implementación de los métodos se hace por otros mecanismos. Esto significa que la creación automática de *código completo* aún no es una realidad.

² El Diseño de Algoritmos Asistido por Computadora (DAAC) es una arquitectura que tiene por objetivo permitir el análisis, diseño, construcción y pruebas de algoritmos a través de un lenguaje visual de modelado que pueda ser utilizado como una agregado a los actuales sistemas CASES con el objetivo de lograr la creación automática de código 3GL completa.

De acuerdo con ANSI/IEEE Std 100-1984, la heurística trata de aquellos métodos o algoritmos exploratorios para la resolución de problemas, en los que las soluciones se descubren por la evaluación del progreso logrado en la búsqueda de un resultado final [51].

En psicología la heurística se relaciona con la creatividad y se ha propuesto que sea aquella regla sencilla y eficiente para orientar la toma de decisiones y para explicar en un plano práctico cómo las personas llegan a un juicio o solucionan un problema.

Combinando estos enfoques se propone la siguiente definición:

Heurística es una regla sencilla y eficiente que pueda orientar en el diseño de algoritmos para la resolución de problemas, en los que los diseños se descubren por la evaluación del progreso logrado en la búsqueda de un algoritmo final.

Dado este análisis, el problema del acercamiento teórico consiste en la identificación de un conjunto pequeño, sencillo y eficiente de heurísticas que, acompañado de una serie de definiciones, conformen un marco teórico conceptual del cual se pueda desarrollar un Modelo de Microingeniería de Software³ para la construcción de algoritmos.

1.2.2. Modelo Humano

Siendo el factor humano un elemento crítico del modelo, es total comprender los procesos mentales a través de los cuales los futuros programadores e ingenieros de software absorben las teorías y desarrollan habilidades y competencias. Actualmente, a nivel mundial existe una fuerte corriente en la formación de profesionistas orientada hacia el desarrollo y evaluación de las competencias profesionales.

En particular, la competencia de la programación es una exigencia obligatoria para la calidad del software. El Software es un producto que se fabrica a partir de teorías y modelos de ingeniería utilizando como maquinaria la mente humana.

El término de competencia laboral inició su auge en la década de los ochentas en países como Inglaterra, Estados Unidos, Australia, Canadá y Francia. En México según el organismo establecido para normar las competencias denominado CONOCER (Consejo de Normalización y Certificación de Competencia [41]) estipula que es la "Capacidad productiva de un individuo que se define y mide en términos de desempeño en un determinado contexto laboral, y no solamente de habilidades, destrezas y actitudes; estas son necesarias pero no suficientes por si mismas para un desempeño efectivo". En España, según el Instituto del Empleo (INEM [92]) se define como el ejercicio eficaz de las capacidades que permiten el desempeño de una ocupación, respecto a los niveles requeridos de empleo "es algo mas que el conocimiento técnico que hace referencia al saber y al saber hacer, El concepto de competencia engloba no solo las capacidades requeridas para el ejercicio de una actividad profesional, sino también un conjunto de comportamientos, facultad de análisis, toma de decisiones entre otros."

³ La Microingeniería de Software en este contexto se refiere a una metodología de análisis, diseño, codificación y pruebas de algoritmos imperativos que residan al interior de los métodos de las clases y que esté basada en las heurísticas y habilidades que el modelo integral contempla.

Partiendo de las definiciones anteriores se extrae que las competencias incluyen, además de conocimientos, también las habilidades, destrezas y actitudes necesarias para poder aplicarlas en el mundo real en la solución de problemas. Estas habilidades se posicionan como aspectos de análisis y toma de decisiones mientras que las destrezas se ubican en las áreas herramientas.

Según el Libro Blanco [67] de la Agencia Nacional de Evaluación de la Calidad y Acreditación (ANECA) de España, el grado de Ingeniería en Informática, que es el equivalente a un Ingeniero en Sistemas Computacionales en México, la programación es una competencia específica de alto impacto en la formación de este profesional.

Paradójicamente, La programación es una de las áreas de mayor índice de reprobación en las Universidades. Solo por citar algunas cifras, en la Universidad del Valle de México (UVM) durante el ciclo escolar 02/05 el total de estudiantes que reprobaron materias en el Departamento de Tecnociencias fue de 144 alumnos de un universo de 520. El número de materias en que se registraron estas no acreditaciones fue de 46 materias. Tan sólo las materias de Lógica de Programación, Estructuras de Datos, programación avanzada y Programación orientada a objetos (5 materias de 46) hay un total de 41 alumnos reprobados, es decir en el 10.86% de las materias se acumula el 28.47% de los reprobados lo cual coloca a estas materias de programación como el área con mayor reprobación del campus. Seguidas en segundo lugar con las materias de Algebra y Ecuaciones diferenciales que ocupan el 2° lugar en reprobación.

Este fenómeno no es exclusivo de la UVM, de acuerdo a diálogos sostenidos al interior de la Mesa Directiva de la Asociación Nacional de Instituciones de Educación en Informática (ANIEI) [74] otras Universidades tanto públicas como privadas observan una tendencia similar con índices de reprobación entre el 30% y el 45% en las áreas de programación e ingeniería de software. A nivel internacional los números no mejoran ya que en las dos Universidades más grandes de España, la Complutense de Madrid y la Politécnica de Cataluña los índices de reprobación oscilan entre el 48% y el 52% [56] y esto se replica al resto de las Universidades Españolas.

Ante esta realidad resulta urgente identificar cuales son las habilidades particulares que conforman la competencia de la programación, realizar una taxonomía de las mismas y posteriormente crear una metodología que desarrolle adecuadamente cada una de estas habilidades apoyadas en un modelo de heurísticas. Identificar y desarrollar las habilidades de la programación es atacar el origen de los problemas y no sus efectos.

1.2.3. Modelo Tecnológico

En la actualidad, como en décadas pasadas, la necesidad del diseño algorítmico sigue presente, sin embargo, las técnicas y herramientas para el modelado algorítmico no han avanzado al mismo ritmo que los lenguajes de modelado de la ingeniería de software de alto nivel.

Las metodologías incrementales, en espiral, basadas en prototipos y sus combinaciones han venido facilitando el tránsito del dominio del problema al dominio de la solución. Aportaciones como el UML [13], y sus agregados posteriores, han facilitado representar con mayor veracidad el mundo real y sus complejas interacciones. Los casos de uso, los contratos, los diagramas de

secuencia, de colaboración, la generalización, la composición, etc. permiten a un arquitecto de sistemas tener una visión muy clara de las clases, su jerarquía y los mensajes que entre ellas fluyen para su colaboración. Sin embargo, en el corazón de las clases, encapsulados dentro de los métodos, sigue siendo necesaria la creación de algoritmos para poder implementar los diversos comportamientos que eventualmente un objeto ejecutará. Es aquí, justamente, dónde las metodologías actuales se separan de las convencionales para el diseño de algoritmos.

UML, por sí mismo, no cuenta con sintaxis para el modelado algorítmico, de hecho, el Teorema de Jacopini (que se comenta en el capítulo dos de este documento) menciona que se deben implementar al menos tres estructuras de control para poder modelar un algoritmo imperativo, estas son la secuencial, la condicional y la cíclica. Los diagramas de UML actualmente no cuentan con la capacidad de representar esa semántica. Incluso el diagrama de actividades, que fue un agregado posterior al UML, tampoco incluye una sintaxis para implementar el teorema de Jacopini. Como consecuencia se tiene que los productos CASE basados en este lenguaje no cuentan con elementos para el modelado algorítmico y están incapacitados para realizar procesos de ingeniería hacia adelante completa, entendiendo ésta como la capacidad de crear código fuente en un lenguaje 3GL a partir del modelado. Si bien muchas herramientas argumentan poder realizar ingeniería hacia adelante, ésta está limitada únicamente a la creación de la interfaz de la clase dejando la sección de implementación a nivel de esqueleto.

Las técnicas tradicionales de modelado algorítmico fueron creadas bajo el paradigma estructurado y filosóficamente guardan tal distancia con las metodologías actuales que complica su uso y operación. Entre las técnicas más famosas se encuentran los diagramas de Warnier, Bertini, Jackson, Chapin, Nassi/Shneiderman, Tabourier, y otros (todos ellos se explican en el capítulo dos). Solo algunos de ellos, conceptualmente hablando, son compatibles con la programación Orientada a Objetos ya que son producto del paradigma estructurado y siguen una estrategia de descomposición funcional.

Por otra parte, las herramientas no han sido actualizadas, entre ellas se encuentran los diagramas de bloques, diagramas de flujo, HIPO, pseudo-código, tablas de verdad, tablas de decisiones, árboles de decisiones, ordinogramas e incluso los grafos (también todas son descritas el capítulo dos). Ninguna de ellas ha sido retomada formalmente y solo unas pocas, como el pseudo-código, continúan vigentes de manera un tanto paralela al proceso de modelado mismo.

Basándose en este análisis se juzga que es necesario contar con una técnica formal de modelado algorítmico que implemente el teorema completo de Jacopini pero que contemple toda la visión y filosofía del modelado Orientado a Objetos; que sea visual pero al mismo tiempo práctico y compacto; que contemple una serie de heurísticas claras y simples de diseño para facilitar el modelado algorítmico; y que pueda ser automatizado en forma de herramienta de software que se complemente con las herramientas CASE actuales;

1.2.4. Definición del Problema

Con la intención de aportar rigor metodológico se exponen los componentes teóricos formales de una investigación científica comenzando con el planteamiento del problema en forma de pregunta relacionando la variable independiente 'X' y la Dependiente 'Y' de la siguiente forma:

¿El Modelo de Construcción de Algoritmos Apoyado en Heurísticas mejora la habilidad humana para diseñar y construir algoritmos imperativos?

1.2.5. Variables Independientes y Dependientes

X: Modelo de Construcción de Algoritmos Apoyado en Heurísticas.

Este modelo permite analizar, diseñar y construir algoritmos imperativos apoyados en heurísticas y está formado por los siguientes componentes:

Elementos Teóricos:

1. Definiciones Formales
2. Heurísticas de Diseño Algorítmico
3. Metodología de Microingeniería de Software

Elementos Humanos:

4. Taxonomía de Habilidades Mentales
5. Metodología de Desarrollo de Habilidades Mentales

Elementos Tecnológicos:

6. Lenguaje Visual de Modelado Algorítmico
7. Arquitectura de un sistema de Diseño de Algoritmos Asistido por Computadora

Y: Habilidad humana para diseñar y construir algoritmos imperativos.

Capacidad humana de poder plantear un problema, comprenderlo, proponer soluciones, diseñarlas y construirlas a través de la elaboración de algoritmos imperativos. Esta habilidad puede ser medida por la puntuación obtenida en una prueba de habilidades de diseño algorítmico

1.2.6. Descripción del Trabajo

Esta investigación está compuesta de dos grandes momentos. El primero es el diseño de un modelo que permita mejorar las habilidades humanas para diseñar y construir algoritmos imperativos basándose en heurísticas. El segundo momento es la comprobación científica de la validez del modelo a través de un experimento de campo.

Un modelo, tal como lo define la enciclopedia Británica [25], es una reproducción ideal y concreta de un objeto o de un fenómeno con fines de estudio y experimentación. La palabra modelo viene del latín *modulus* que significa molde. Existen diversos tipos de modelos, entre ellos se encuentran los matemáticos, de estructuras, de máquinas, de procesos industriales, económicos, de fenómenos físicos, etc. Los modelos se utilizan para diferentes cosas, por ejemplo cuando resulta demasiado complicado o incierto un estudio analítico, o cuando un

estudio experimental directo presenta dificultades de carácter económico y prácticos difíciles de superar. Un modelo, dependiendo de su naturaleza, estará formado por un conjunto de componentes y reglas que rigen su interacción.

Este trabajo aspira a construir un modelo que aporte una serie de elementos y reglas para facilitar el modelado de algoritmos utilizando las teorías de programación que las ciencias de la computación han aportado a través de los años.

Este modelo contendrá los siguientes elementos:

Del Modelo Teórico:

- Un conjunto de principios teóricos y definiciones formales que aporten consistencia y solidez al modelo.
- Un conjunto compacto y práctico de heurísticas que sean aplicables a través de la microingeniería de software para conducir el modelado y diseño de algoritmos imperativos.
- Una metodología de microingeniería de software que aporte claridad en su proceso de implementación y operación.

Del Modelo Humano:

- Una taxonomía de habilidades mentales que inciden directamente en la competencia de la programación
- Una metodología de desarrollo de habilidades que se convierta en guía de cómo desarrollar el talento de las personas en forma iterativa e incremental.

Del Modelo Técnico:

- Un lenguaje de modelado visual algorítmico para diseñar algoritmos con una gramática que considere sintaxis y semántica para implementar desde el teorema de Jacopini hasta los principios de herencia, polimorfismo y encapsulamiento, así como otros que se consideren convenientes. Este lenguaje tendrá una serie de cualidades específicas para facilitar el modelado retomando las virtudes y evitando las desventajas que diversos modelos utilizaron a través de los años. Este lenguaje inicialmente servirá de base para la creación de código 3GL pero, eventualmente, podría ser compilado directamente.
- Una arquitectura que permitirá la eventual construcción de herramientas de software bajo el concepto de Diseño de Algoritmos Asistido por Computadora. Esta arquitectura permitirá a las herramientas de software de este tipo interactuar, como agregados (Add on's), con las actuales herramientas CASE permitiendo la generación de código 3GL de forma automática bajo el principio de Ingeniería Hacia Adelante Completa. Este principio consiste en aportar la capacidad de crear el código 3GL en la sección de implementación de las clases, dentro de los métodos. Esto no sucede hoy en día.

Estos elementos integran el Modelo de Construcción de Algoritmos Apoyado en Heurísticas que se desea desarrollar.

Una vez diseñado el modelo se aplicará una investigación de campo que compruebe experimentalmente que en verdad mejora la habilidad humana para diseñar algoritmos. Se obtendrán datos de pruebas de habilidades antes y después de la aplicación contemplando un grupo de control y uno de aplicación y se analizarán los resultados para conocer si existe diferencia estadísticamente significativa entre la aplicación del modelo y la no aplicación del mismo. Se instrumentarán estrategias específicas de control de las variables extrañas para minimizar su impacto y poder llegar a conclusiones sobre la validez del modelo.

1.2.7. Aportaciones, Alcances y Limitaciones

Las aportaciones, alcances y limitaciones definen la frontera del trabajo que se realizará en función de la solución al problema expuesto anteriormente.

1.2.7.1. Aportaciones y Beneficios

El desarrollo de un modelo que se aspira integrar contemplando los tres enfoques ya citados permitirá los siguientes beneficios:

1. Contar con un marco teórico-metodológico para diseñar algoritmos a través de heurísticas prácticas y simples.
2. La microingeniería de software permitirá ampliar las fronteras de los actuales sistemas CASE permitiendo la construcción de código 3GL completo.
3. Contar con una taxonomía de las habilidades a través de la cual se pueda identificar las fortalezas y debilidades de un futuro ingeniero y crear un plan de desarrollo y consolidación de las mismas.
4. Mejorar las habilidades de programación de los estudiantes de esta profesión disminuyendo los índices de reprobación y creando talento intelectual que incida en la calidad del software.
5. Contar con un modelo tecnológico para poder desarrollar futuras herramientas automatizadas de software que automaticen la codificación al interior de los métodos de las clases a través de un proceso formal de diseño de algoritmos evitando que se pase del análisis a la codificación directa.
6. Crear los primeros modelos para la creación de compiladores de lenguajes visuales a nivel de 3GL.

1.2.7.2. Alcances

El alcance particular al problema expuesto es presentar:

Desde el enfoque teórico:

- Un conjunto de definiciones formales que conformen un marco teórico conceptual para el modelo.
- La propuesta de un conjunto de heurísticas claras para ser aplicadas en el proceso de diseño algorítmico utilizando el lenguaje propuesto.
- El diseño de una metodología de microingeniería de software.

Desde el enfoque humano:

- El diseño de una taxonomía de habilidades mentales superiores que identifiquen en lo específico como fortalecer en forma iterativa e incremental la habilidad humana del diseño de algoritmos imperativos.
- Una metodología de desarrollo de habilidades humanas

Desde el enfoque técnico:

- El diseño de un lenguaje visual de modelado algorítmico que contemple las teorías de programación vigentes y con aspiración de algunas futuras.
- Una arquitectura para la construcción de un sistema de Diseño de Algoritmos Asistido por computadora.

1.2.7.3. Limitaciones

Este trabajo se limita a desarrollar los puntos expuestos en la sección de alcances del presente documento y no ambiciona más que definir claramente un modelo teórico conceptual probado experimentalmente que pueda, eventualmente, convertirse en una plataforma de desarrollo de tecnologías de software.

1.2.8. Justificación

Diversas son las justificaciones para la elaboración de este trabajo. Serán abordadas tres perspectivas: la social, la profesional y la científica para sustentar la valía de un trabajo de esta naturaleza.

1.2.8.1. Social

Las ciencias de la computación, y en particular los sistemas de computadora, son elementos cuyo objetivo es potenciar las capacidades del ser humano en cada uno de los ámbitos en que éste incursione, ya sea la medicina, las finanzas, el comercio, la educación, el ejercicio de la justicia o la exploración del espacio. A través de los sistemas de computadora las tareas se simplifican y agilizan lo que permite hacer más cosas en menos tiempo y con mayor precisión y confianza. Sin embargo, el desarrollo de sistemas de computadora está aún lejos de ser una actividad libre de tropiezos, retardos, sobre-costos y frustraciones. Año tras año miles de proyectos de software son iniciados con alcances y presupuestos claramente establecidos, sin embargo, solo unos pocos serán concluidos en el tiempo y costo programado; otros terminarán con retrasos y presupuesto excedido; varios serán recortadas sus expectativas para poder concluirlos; algunos más serán

abandonados en el camino considerándose pérdida total; y, finalmente, otros pocos serán tan catastróficos que pondrán en jaque la supervivencia misma de la institución que los impulsó. Ante este escenario todos los esfuerzos que se hacen en mejorar las áreas de ingeniería y calidad de software son de alto impacto para la sociedad a la que se sirve. La historia de la computación ha demostrado que la innovación en metodologías para incrementar la certidumbre de un proyecto de software concluye en mejores prácticas de ingeniería.

Por otro lado, la educación y desarrollo del talento humano, específicamente en lo que a habilidades de solución de problemas a través de algoritmos imperativos se refiere, inciden directamente en la perpetua misión de aprender, trascender y mejorar la calidad de vida de todos los miembros de nuestra especie.

Concretamente este trabajo aspira a proponer un modelo de ingeniería de software concentrado en la parte microscópica del proyecto pero que al mismo tiempo es el corazón del mismo: los algoritmos. Siendo este elemento una pieza trascendental para la calidad y cumplimiento de las metas del proyecto y no habiendo modelos formales vigentes apegados a las teorías actuales para su diseño es claro que realizar esfuerzos en esta línea está, por demás, justificado.

1.2.8.2. Profesional

A través de los años se ha observado como de forma consistente y progresiva muchos procesos evolucionan hacia modelados visuales. Dentro de los ejemplos más representativos se encuentran las implementaciones de Bases de Datos Relacionales que utilizan el modelo de Codd. Antiguamente, los modeladores de bases de datos dibujaban los diagramas aplicando las reglas del modelo relacional en algún formato (electrónico o no) para después escribir el código en lenguaje 4GL que construye y define las tablas, índices, relaciones y reglas. Hoy día, a través de herramientas visuales, el modelado es prácticamente la única tarea que se debe de hacer ya que con el uso de herramientas computacionales se puede crear el código 4GL simplemente utilizando el “botón” de ingeniería hacia adelante ahorrando importante tiempo al diseñador. Esto mismo sucede cuando se trata de crear una vista, un query u otros elementos. Basta con modelarlos visualmente y haciendo ingeniería hacia adelante se contará en unos pocos segundos del código necesario para su ejecución.

La microingeniería de software es un modelo que permitirá crear código completo 3GL dentro de la sección de implementación de las clases a través de técnicas de ingeniería hacia adelante haciendo uso de un lenguaje de modelado visual y un conjunto de heurísticas. Esta es una necesidad que no ha sido satisfecha por los sistemas CASE actuales a causa de la inexistencia de un modelo formal que lo sustente y proporcione una plataforma sólida para su creación y desarrollo.

Al contar con esta funcionalidad los sistemas de Diseño de Algoritmos Asistidos por Computadora serán una realidad y podrán operar de forma combinada con los actuales sistemas de Ingeniería de Software complementándolos y ampliando sus fronteras hasta dónde no habían llegado: la implementación de los métodos de las clases en forma fácil y automática.

La existencia de un modelo con las características que se aspira incluir constituiría una plataforma para el diseño, modelado y evaluación de algoritmos moderna incluyendo las teorías

vigentes y previendo la futura aparición de nuevas. Los ingenieros de software profesionales se verán beneficiados con el nacimiento de herramientas computacionales que automatizarán procesos que hoy día se hacen a mano y, en muchas ocasiones, con diseños parciales o directamente sobre la computadora sin diseño específico preliminar.

1.2.8.3. Científica

Dentro de los alcances de este trabajo esta la constitución de una arquitectura para la creación de herramientas de Diseño de Algoritmos Asistido por Computadora lo que permitirá a los ingenieros de software y programadores modelar los algoritmos internos a los métodos de las clases logrando la creación automática de código 3GL. Sin embargo, ese código aún tiene que ser compilado para su ejecución.

El nacimiento de este modelo permitirá la futura abolición de los lenguajes 3GL, es decir, en los próximos años una herramienta de modelado puede realizar el diseño de los algoritmos de forma visual y directamente de ese lenguaje compilarlo y llevarlo a código máquina (o paquetes de bytes si es lo que se desea) sin pasar por el lenguaje 3GL. Los IDE's de programación en C++, Java o Delphi podrán ser herramientas de modelado algorítmico trabajando con un lenguaje de Generación Visual que incluya un compilador, un depurador, un editor de interfaces de usuario y el resto de los elementos que se acostumbran.

El futuro podrá estar basado en compiladores de Lenguajes Visuales en lugar de compiladores 3GL. Este modelo sembrará la semilla para ese futuro.

Adicionalmente, el diseño experimental y pruebas de campo controladas aportarán validez científica formal al modelo y crearán una plataforma sólida para investigaciones sobre este particular.

1.3 Hipótesis

Este trabajo propone una hipótesis de tipo descriptiva que relaciona dos variables en forma de asociación o covarianza en términos de dependencia.

Ho (Hipótesis nula): El modelo de Construcción de Algoritmos Apoyado en Heurísticas no mejora las habilidades humanas para diseñar y construir algoritmos imperativos.

Hi (Hipótesis alternativa): El modelo de Construcción de Algoritmos Apoyado en Heurísticas mejora las habilidades humanas para diseñar y construir algoritmos imperativos.

Esta hipótesis busca explicar y predecir el comportamiento humano al aplicar el modelo, por lo tanto implica el control de la variable independiente para ver el efecto que produce en la dependiente eliminando y controlando las variables extrañas.

1.4 Objetivos

1.4.1. Principales

1. Crear un modelo de Construcción de Algoritmos Apoyado en Heurísticas que contemple de forma integral el trípode de la programación para que mejore la habilidad humana en el diseño y construcción de algoritmos imperativos.
2. Diseñar y aplicar un experimento de campo que pruebe que efectivamente el modelo de Construcción de Algoritmos Apoyado en Heurísticas mejora la habilidad humana para el diseño y construcción de algoritmos imperativos.

1.4.2. Específicos

1. Diseñar una metodología de desarrollo de habilidades de programación.
2. Diseñar una metodología de microingeniería de software.
3. Diseñar un lenguaje visual de modelado imperativo.
4. Crear un conjunto de heurísticas de diseño algorítmico.
5. Diseñar una arquitectura para la creación de sistemas de Diseño de Algoritmos Asistido por Computadora que pueda extender las capacidades de los CASE actuales.

1.5 Método

El presente documento, de acuerdo a su finalidad, es una Investigación de tipo aplicada porque busca generar conocimientos para resolver un problema específico de acuerdo a un objetivo práctico.

De acuerdo a su alcance es de tipo comparativo y predictivo. Comparativo porque establece correlaciones entre variables para formular y probar hipótesis de tipo causal entre el antes y después de la aplicación de la metodología además de ser ejecutado con grupos de investigación experimental y de control. Es de tipo predictivo porque al comprobar las correlaciones causales entre las variables pretende predecir el comportamiento humano en términos de la mejora en sus habilidades para diseñar y construir algoritmos.

La investigación se realizará en dos momentos. En el primero (Diseño del Modelo de Construcción de Algoritmos Apoyado en Heurísticas) se aplicará el método Teórico Deductivo y en el segundo (para corroborar el modelo) el método Experimental. El método Teórico Deductivo se utilizará porque se basa en el razonamiento y comprende un momento deductivo y un momento inductivo para el diseño y construcción del Modelo de Construcción de Algoritmos

Apoyado en Heurísticas. Posteriormente se hará uso del método experimental porque se diseñará y aplicará un experimento de campo para corroborar las hipótesis planteadas como resultado del diseño del Modelo.

Como consecuencia de que la segunda parte de esta investigación es de tipo experimental de campo es necesario definir la población, muestra, variables, diseño del experimento, técnicas de control de variables extrañas, instrumentos, escenarios y procedimiento. A continuación se explican estos elementos que son parte integral del método de investigación.

1.5.1. Población y Muestra

Por población se entiende el conjunto de personas que cubren los criterios para poder participar en el experimento de campo. En este estudio representa al universo total. La muestra representa al subconjunto del universo que será elegido para participar.

Población: Estudiantes de primer ingreso a las carreras de Ingeniería sin experiencia previa en programación de la Universidad del Valle de México, Campus Hispano, que llevarán la materia de Principios de Programación.

Muestra: Los dos primeros grupos que se formen resultado de la inscripción en la materia Principios de Programación para el ciclo de clases que inicia en agosto de 2005 y termina en diciembre del mismo año y que muestren que no tienen diferencias significativas en sus antecedentes de habilidades.

1.5.2. Definición de Variables

En el capítulo tres se presentan, como parte del modelo, una serie de definiciones formales de cada uno de los conceptos utilizados en este trabajo. Para fines prácticos de esta sección serán definidas únicamente las variables que forman parte del diseño del experimento y control de variables extrañas.

Definición Operacional:

- Yb: Valor promedio ponderado obtenido del conjunto de pruebas aplicadas al inicio del curso tanto por el grupo de investigación como por el grupo de control antes de la aplicación de la metodología para inducir el Modelo de Construcción de Algoritmos Apoyado en Heurísticas a la muestra. Es la variable dependiente antes de X. Se utiliza para valorar la homogeneidad de los grupos al iniciar el experimento.
- X: Representa la aplicación de la Metodología para inducir el Modelo de Construcción de Algoritmos Apoyado en Heurísticas en el grupo investigado. Es la variable Independiente.
- ~X: Representa la NO aplicación de la Metodología para inducir el Modelo de Construcción de Algoritmos Apoyado en Heurísticas en el grupo de control.
- Ya: Valor promedio obtenido en la prueba de habilidades de diseño algorítmico tanto por el grupo de investigación como por el grupo de control después de la

aplicación de la metodología para inducir el Modelo de Construcción de Algoritmos Apoyado en Heurísticas a la muestra. Es la variable dependiente después de X. Se utiliza para valorar las diferencias obtenidas resultado de aplicar, o no X y aceptar o rechazar la hipótesis.

- Mr: Representa que los grupos muestra están homogenizados y son aleatorios.
- Ge: Grupo Experimental. Representa al grupo en el cual se aplicará la metodología de inducción al Modelo de Construcción de Algoritmos Apoyado en Heurísticas.
- Gc: Grupo de Control. Representa al grupo que desarrollará sus actividades normales sin la aplicación del Modelo de Construcción de Algoritmos Apoyado en Heurísticas.
- Ho: Hipótesis Nula.
- Hi: Hipótesis Alternativa
- Be: Porcentaje de Bajas del Grupo Experimental
- Bc: Porcentaje de Bajas del Grupo de Control.

1.5.3. Diseño de Investigación

El presente experimento se diseñó utilizando dos grupos, uno de aplicación (Ge) y uno de control (Gc). Ambos grupos se consideran homogéneos y aleatorios (Mr). Se aplica preprueba (Yb) y posprueba (Ya) a ambos grupos en dónde uno de ellos es sometido al estímulo del Modelo de Construcción de Algoritmos Apoyado en Heurísticas (X) y el otro no (~X).

	Yb	X	Ya (Ge)
Mr -----	Yb	~X	Ya (Gc)

El experimento durará cuatro meses (el equivalente de un cuatrimestre o ciclo lectivo en la Universidad del Valle de México Campus Hispano). Durante este período tanto Ge como Gc cursarán la Materia de Principios de Programación pero en el caso de Ge, además, se le introducirá al Modelo de Construcción de Algoritmos Apoyado en Heurísticas. Al inicio del experimento se aplicará una batería de pruebas de habilidades y encuestas de experiencia previa (Yb) para poder determinar la validez de la aleatoriedad y homogeneidad entre Ge y Gc. Al finalizar el Experimento se aplicará una batería de pruebas de habilidades (Ya) para determinar el avance comparativo entre el antes y después tanto de Ge como de Gc.

Durante el experimento se harán pruebas comparativas entre antes y después para cada grupo además de entre ellos mismos.

Se concluirá que Hi es cierta y se descartara Ho si el Ya de Ge tiene diferencia estadística significativamente mayor que el Ya de Gc.

Se aceptará Ho como cierta y se descartará Hi si el Ya de Ge No tiene diferencia estadística significativamente mayor que el Ya de Gc.

1.5.4. Control de Variables Extrañas

Un aspecto fundamental que debe ser considerado para conservar la validez y confiabilidad del experimento es el control de las variables extrañas o externas ya que se requiere eliminar su influencia en la variable dependiente. A continuación se citan las variables extrañas y las estrategias para evitar su influencia y alteración de resultados:

Antecedentes: Se refiere a los antecedentes de habilidades que los integrantes de los grupos puedan tener antes de iniciar el experimento. Si existen individuos pertenecientes a los grupos Ge y Gc que ya tengan experiencia previa en diseño de algoritmos y programación alterarían los resultados. Para controlar esta variable se han elegido grupos de primer ingreso con la esperanza de que la experiencia previa sea mínima; se aplicarán prepruebas de habilidades a cada miembro del grupo, sus resultados serán promediados (Yb) y serán comparados entre Ge y Gc buscando que no haya diferencias estadísticamente significativas. Si las hubiera no se considerarían adecuados para el experimento y se seleccionará otro grupo para muestra.

Maduración: Se refiere al desarrollo natural en sus habilidades que tendrán tanto Ge como Gc durante el transcurso del estudio de su materia de lógica de programación. Para controlarlo se aplica preprueba (Yb) y posprueba (Ya) a ambos grupos y se espera naturalmente que el Ya de ambos grupos sea mayor que su Yb pero la aceptación o rechazo de Hi dependerá de la comparación del Ya entre Ge y Gc y no del cambio entre Ya y Yb de cada grupo.

Bajas: Se refiere a la mortalidad experimental, es decir al porcentaje de alumnos que en el transcurso del ciclo escolar tramitan su baja o abandonan la materia. Para controlarlo se medirá Be y Bc al finalizar el período escolar y se compararán entre ellos. Se considerará que el experimento es aceptable si la diferencia entre ellos es menor al 10%.

Instrumentación: Se refiere a los cambios en los instrumentos de medición entre preprueba y posprueba. En este caso particular la aceptación o rechazo de Hi depende de la comparación entre los resultados de Ya entre Ge y Gc y no por la comparación entre Ya y Yb. Por lo tanto el control de instrumentación consistirá en que la batería de pruebas aplicadas en Yb (tanto en Ge como Gc) sea el mismo y bajo las mismas condiciones. De igual forma, la Batería de pruebas de Ya (tanto en Ge como en Gc) también será la misma.

Reactividad al estímulo: Se refiere al proceso mismo del desarrollo de la materia de Principios de Programación. Este estímulo puede ser afectado por elementos tales como la capacidad del profesor que imparte la materia, disponibilidad de equipo y herramienta para su ejercicio, así como bibliografía y otros materiales. Para su control se utilizarán profesores con un mismo nivel de desempeño en el Sistema de Evaluación de Profesores de la UVM, se realizarán las prácticas con las mismas herramientas de hardware y software y se dispondrá de los mismos recursos bibliográficos y materiales.

1.5.5. Instrumentos

Los instrumentos principales para realizar el experimento son:

- El Modelo de Construcción de Algoritmos Apoyado en Heurísticas que es resultado del trabajo de la primera etapa de esta investigación.
- La batería de pruebas de habilidades que se aplicarán como preprueba y posprueba.
- Las tablas de análisis estadístico de correlación de variables para encontrar diferencias significativas entre medias de resultados.
- Documentos complementarios tales como control de asistencia y puntualidad de profesores; resultados de evaluación y desempeño de profesores; actas de calificaciones parciales y finales; control de bajas de estudiantes; temarios, bibliografía y paquetes de practicas de la materia; etc.

1.5.6. Escenario

El experimento se realizará en el Campus Hispano de la Universidad del Valle de México ubicado en el 222 de Av. José López Portillo en Coacalco de Berriozabal, Estado de México. Los escenarios básicos en los que trabajarán los grupos experimental y de control son los siguientes:

1. Aulas de clase con capacidad hasta para cincuenta personas equipadas con mesa bancos, pizarrones blancos y ventilación e iluminación adecuadas.
2. Laboratorios de cómputo equipados con computadoras Compaq y arquitectura Pentium IV de 2.2 Ghz, 256 MRAM de Memoria, Discos duros de 80 GBytes, todas ellas conectadas en red y con acceso a Internet y otros servicios tales como impresoras, scanners, quemadores de discos compactos y dispositivos multimedia.
3. Diversos lenguajes de programación y otros softwares de apoyo.
4. Biblioteca con servicios bibliográficos, hemerográficos y de biblioteca electrónica con acceso a diversas bases de datos.
5. Salas de usos múltiples con cañones instalados.
6. Patios, canchas deportivas y otros escenarios de recreación y esparcimiento.

1.5.7. Procedimiento

Para la elaboración del modelo de Construcción de Algoritmos Apoyado en Heurísticas:

1. Se realizará una investigación del estado del arte en teorías de programación, paradigmas y metodologías desde la perspectiva del modelado algorítmico.
2. Se realizará una investigación del estado del arte en técnicas, herramientas y métodos de modelado algorítmico al interior del seno de cada paradigma y teoría de programación.
3. Se realizará una investigación del estado del arte sobre modelos y enfoques sobre la educación, enseñanza, aprendizaje y desarrollo de habilidades para el modelado algorítmico.
4. A la luz de la información recabada se diseñarán y crearán los siguientes elementos del modelo de Construcción de Algoritmos Apoyado en Heurísticas:
 - A. Definiciones formales de sus componentes.
 - B. Heurísticas de diseño

- C. Metodología de Microingeniería de Software
 - D. Taxonomía de habilidades
 - E. Modelo para el desarrollo de habilidades
 - F. Lenguaje visual de modelado
 - G. Arquitectura para un sistema de Diseño de Algoritmos Asistido por Computadora.
5. Se aplicará el experimento de campo tal como se explica en los siguientes párrafos.

Para la aplicación del experimento de campo:

1. Se identificarán las pruebas existentes de habilidades en el diseño de algoritmos, se analizarán y se seleccionará aquella que mida de la mejor forma posible el conjunto de habilidades identificadas en la taxonomía de habilidades del Modelo.
2. Se identificarán los grupos de alumnos que se han matriculado a la materia de Principios de Programación que se imparte en el primer cuatrimestre de las carreras de Ingeniería.
3. Se aplicarán las prepruebas de habilidades a los grupos registrados para verificar sus antecedentes, homogeneidad y aleatoriedad eligiéndose aquellos dos cuyas diferencias estadísticas no sean significativas.
4. Se elegirá al azar uno de ellos para la aplicación del Modelo.
5. Se elegirán profesores que tengan un equivalente nivel de desempeño y éxito según resultados de las pruebas de evaluación docente de la UVM.
6. Los profesores desarrollarán a lo largo del cuatrimestre su clase de Principios de Programación en forma normal cubriendo el mismo temario y utilizando la misma bibliografía, mismo paquete de prácticas, mismo equipo de laboratorio y mismo conjunto de recursos audiovisuales y de infraestructura.
7. Al grupo experimental, además de la temática de su materia, se le introducirá al Modelo de Construcción de Algoritmos Apoyado en Heurísticas.
8. Al terminar el ciclo se aplicará a ambos grupos la batería de posprueba de habilidades incluyendo una específica de diseño y construcción de algoritmos.
9. Se valida el control de variables extrañas.
10. Se obtendrán los resultados y se realizarán los análisis correspondientes para determinar si se acepta o se rechaza la hipótesis nula y alternativa.
11. Redactar las conclusiones y hallazgos.

Capítulo 2. Marco Teórico: Reporte del Estado del Arte

El diseño de algoritmos es una parte crítica en el proceso de ingeniería de software. A lo largo de los años han surgido diversos enfoques, técnicas y herramientas para su diseño y construcción. En este capítulo se recopilan las principales metodologías de diseño algorítmico desde 1955 a la fecha.

La programación sigue siendo el corazón de todo proyecto de software. Este capítulo, además, hace una recopilación histórica desde 1945 a la fecha de las principales teorías de programación y como estas han dictado la creación de nuevos lenguajes, paradigmas, técnicas y herramientas que eventualmente se convierten en los tópicos curriculares de los centros educativos.

La formación de recursos humanos competentes y hábiles para diseñar sistemas, algoritmos y programas no es solamente una cuestión de temas dentro de un plan de estudio. Estableciendo una analogía con la construcción de software, es tan importante el producto terminado como el proceso a través del cual se formó. En consecuencia, este capítulo también incluye una recopilación, desde 1960 a la fecha, de las diversas técnicas de enseñanza que se han utilizado.

2.1 Teoría de la Programación

En todas las ciencias, y esta obviamente no es la excepción, primero han surgido teorías de cómo deben hacerse las cosas para posteriormente llevarlas a la práctica y corroborar si son correctas o no. Esta sección describirá las teorías de la programación que, si bien no se encuentran absolutamente todas, se retoman las que has sido más aceptadas por la industria y han marcado hitos históricos en ésta área del conocimiento.

2.1.1. Conceptos

La teoría de la programación será considerada como el conjunto de principios que rigen un paradigma de programación y definen las características semánticas de cierto conjunto de lenguajes.

Algunos autores realizan una diferenciación entre el concepto de programar y el de codificar Levine [66] señala lo siguiente: “Por programar se entiende un proceso mental complejo, dividido en varias etapas. La finalidad de la programación, así entendida, es comprender con claridad el problema que va a resolverse o simularse por medio de la computadora, y entender también con detalle cuál será el procedimiento mediante el cual la máquina llegará a la solución deseada.

“La codificación constituye una etapa necesariamente posterior a la programación, y consiste en describir, en el lenguaje de programación adecuado, la solución ya encontrada, o sugerida, por medio de la programación. Es decir, primero se programa la solución de un problema y después hay que traducirla a la computadora.”

Para efectos de este trabajo se utilizará un concepto de programación amplio basado en el planteamiento de Levine que incluye el proceso de identificación del problema, el diseño algorítmico de la solución, la codificación en algún lenguaje y las pruebas correspondientes.

La teoría de la programación aspira a dar certidumbre sobre las formas más adecuadas de enfrentar un problema, modelarlo y encontrar su solución algorítmica para poder codificarlo en algún lenguaje y ejecutar la solución en una computadora. Para ello ha creado, con el paso del tiempo, una serie de principios que han venido evolucionando. Estos principios han dado nacimiento a diversos paradigmas dentro de los cuales han surgido lenguajes de programación, técnicas específicas de diseño, herramientas (tanto computacionales como manuales) para ayudar durante el proceso de solución de problemas. Desde el nacimiento de las computadoras han pasado poco más de cinco décadas pero únicamente ha habido unos cuantos hitos que han marcado la historia. Estos son los que serán mencionados.

2.1.2. La Programación Como un Arte

El corazón de todo producto de software es su código. Código que está escrito en algún lenguaje. Código que está apegado, en mayor o menor medida, a un paradigma regido por sus propias reglas. Código que fue escrito por personas con distintos talentos. Código resultado de un proceso de análisis y diseño algorítmico. Código obtenido como conclusión de un esfuerzo de programación. Programación realizada, en parte, utilizando técnicas, teorías, reglas, principios y, por otra parte, una buena dosis de inspiración sublime prácticamente artística

Importantes autores siguen considerando que la programación aún no está en condiciones de recibir el título de ciencia. Budd [17] señala “Es cierto que programar un computador es todavía una de las tareas más difíciles jamás emprendidas por la humanidad; llegar a ser un experto en programación requiere talento, creatividad, inteligencia, lógica, habilidad para construir y usar abstracciones, y experiencia, aun cuando se disponga de las mejores herramientas.” Por otra parte Glenn [45] comenta “El proceso de descubrimiento de algoritmos y de resolución de problemas tienen una íntima relación. En última instancia, nos gustaría reducir el mismo proceso de resolución de problemas a un algoritmo, pero se ha demostrado que esto es imposible. Así, la capacidad para resolver problemas sigue siendo más una aptitud artística que debe desarrollarse, que una ciencia precisa por aprender.” Finalmente, Kunth [61], quizás uno de los programadores más notables, nombró a su libro el Arte de la Programación subrayando que a pesar de todos los formalismos que se han creado, el diseño de algoritmos y su codificación sigue siendo un arte.

El valioso acervo de conocimientos, técnicas, herramientas y métodos son imprescindibles para formar a un ingeniero de sistemas. Son el equivalente en los pintores a dominar la teoría del color y las técnicas pictóricas, pero ese conocimiento no puede, ni podrá jamás, guiar su mano para crear una obra maestra.

2.1.3. Evolución de la Teoría de la Programación

Los principios y teorías de la programación han evolucionado constantemente desde los tiempos de la programación dura hasta la era del cómputo distribuido. Cada una de las ideas de

grandes protagonistas de esta joven ciencia ha marcado la historia dejando su huella en miles de líneas de código escrito en todo el mundo. Los principales hitos históricos son los siguientes:

La primera teoría de la programación.- La programación dura. Surge con el nacimiento de la computadora cuando Charles Babbage [65] alrededor de 1830 desarrolla el concepto de La Máquina Analítica. Su arquitectura está formada por cinco módulos: entrada, cálculo, control, memoria y salida. Sin embargo, no fue sino poco más de cien años después (1947) que fue posible la programación gracias a la creación de la ENIAC [65], considerada la primer computadora digital de la historia. La técnica de programación se hacía en forma “dura” significando que estaba físicamente cableada (alambrada dicen algunos) para resolver una tarea específica y si se deseaba tener un programa diferente se tenía que modificar ese cableado. No parece ser necesario justificar la importancia de este hito histórico.

La segunda teoría de programación.- La programación suave. Nace del modelo de John von Neumann [65] y consiste en permitir la coexistencia de datos con instrucciones en la memoria para que la computadora pueda ser programada de forma “suave” y no con cables. Esta idea es de tal envergadura que obliga a los fabricantes de computadoras a revisar su arquitectura. Estas ideas dieron nacimiento al paradigma de programación libre y a los primeros lenguajes de programación.

La tercera teoría de programación.- Teorema de la Estructura. Resultado del trabajo de Conrado Böhm y Giuseppe Jacopini [12]. En ella se presenta el teorema de la estructura y las siguientes definiciones como lo retoma Alcalde en [3]:

- ✓ Diagrama Propio: Aquel que posee un solo punto de inicio y uno solo de fin.
- ✓ Programa Propio: Posee un solo inicio y un solo fin; Todo elemento del programa es accesible, es decir, existe al menos un camino desde el inicio al fin que pasa a través de él; no posee ciclos infinitos.
- ✓ Equivalencia de programas: Dos programas son equivalentes si realizan, ante cualquier situación de datos, el mismo trabajo pero de distinta forma.
- ✓ Teorema de la estructura: Todo programa propio, realice el trabajo que realice, tiene al menos un programa propio equivalente que únicamente utiliza las estructuras básicas de programación, que son: la secuencia, la selección y la repetición.

Este teorema permitió el nacimiento del paradigma de la programación estructurada cuando Dijkstra compiló sus Notes on Structured Programming para luego ser integrado en el libro Structured Programming [77] cuyo coautor es Hoare.

La cuarta teoría de la programación.- ADT, ocultación y acceso uniforme. Se debe al trabajo e ideas de varios autores, entre ellos: Geschke en [44] con el principio de acceso uniforme que proviene desde 1975 cuyo nombre original era “referencia uniforme”; Parnas en [78] con el principio de ocultación de información quien siembra la idea que posteriormente es cosechada bajo el principio de encapsulamiento; y Hoare en [52] y por Liskov y Zilles en [68] con el concepto de Tipo de Dato Abstracto. Con estos elementos y el concepto de Clase que apareció en Simula en 1967 [9] ya todo estaba dado para el nacimiento del paradigma de la programación orientada a objetos.

La quinta teoría de la programación.- Cómputo distribuido. Hecho realidad hoy por estándares como CORBA, COM y DCOM. El paradigma de cómputo distribuido es el resultado de la integración de redes de área dispersa que pueden compartir componentes de software distribuidos geográficamente y que trabajan en colaboración. No es necesario abundar más para los fines de este trabajo.

2.1.4. Los Paradigmas de Programación

Estos hitos históricos señalados en la sección anterior, mismos que incluyen las teorías ya expuestas, dieron nacimiento a filosofías distintas para modelar el mundo real. Estos principios (salvo los dos primeros) en realidad no marcan el nacimiento de una tecnología revolucionaria que haya modificado completamente la forma como se programa. De hecho, el modelo de programación imperativo prácticamente ha sufrido muy pocas modificaciones desde el nacimiento de los lenguajes de alto nivel. Los verdaderos e importantes cambios están más del lado filosófico: cómo transitar del dominio del problema al dominio de la solución.

Existen dos famosos artículos que prueban lo anterior. En 1986 Fredrick P. Brooks, escribió el artículo "No silver bullet" [15], en él apuntaba que en los últimos diez años no se había producido ningún progreso significativo en el desarrollo de software, y analizaba críticamente todas las tecnologías más prometedoras del momento. En respuesta al artículo de Brooks, Brad Cox, el inventor de Objective C, publicó otro artículo: "There is a silver bullet", [29] en el que esencialmente rebatía la tesis de Brooks diciendo: "Existe una bala de plata. ... La bala de plata es un cambio cultural en lugar de un cambio tecnológico. Es un nuevo paradigma; una revolución industrial basada en partes reutilizables e intercambiables que modificará el universo del software" refiriéndose específicamente a la programación Orientada a Objetos. Sin embargo, esta observación es igualmente aplicable al nacimiento de los otros paradigmas.

Paradigma proviene del vocablo griego *paradeigma* (*παράδειγμα*) que significa mostrar o manifestar. Se ha utilizado como conjunto de reglas y principios que rigen una forma de pensar y de actuar. Los paradigmas no son conceptos propiamente del mundo de la computación, más bien éste los ha adoptado para poder expresar sus propias formas, por supuesto, de pensamiento y acción. Siendo un paradigma un aspecto de corte filosófico más que tecnológico no tiene fronteras fijas de nacimiento y muerte, de hecho, a lo largo del tiempo se ha venido solapando un paradigma sobre otro cambiando paulatinamente, más en un proceso evolutivo que revolucionario.

Cada uno de los principales Paradigmas cuenta con su propio conjunto de reglas y principios gestados a partir de las teorías de programación ya expuestas. Se describen a continuación:

2.1.4.1. Paradigma Duro

Este paradigma en realidad es previo al nacimiento del software. Su característica principal es el alambrado físico por medio de cables para la definición del algoritmo. No existen reglas ni técnicas de diseño algorítmico formal. Lo único que hay son datos organizados en tipos primitivos.

Es importante señalar que la fabricación de computadoras en esta temprana etapa se realizaba de forma limitada y con fines principalmente de investigación.

2.1.4.2. Paradigma Libre

Es producto del concepto de John von Neumann de programa almacenado. Además de los datos y tipos primitivos de la programación dura se incluyeron estructuras de datos más complejas tanto escalares (registros, arreglos, etc.) como dinámicas (pilas, colas, listas, etc.) así como el nacimiento de los primeros algoritmos de administración de comportamiento de estas estructuras de datos. No se cuenta con métodos formales de diseño algorítmico ni de sistemas pero en los lenguajes se implementan las siguientes estructuras básicas de control: secuencia, repetición, salto condicional y salto incondicional. Durante esta época hay un auge en el nacimiento de lenguajes que van desde el ensamblador y el macro ensamblador a los primeros lenguaje 3GL como Fortran, Cobol, Algol, Lisp, PL/1, RPG, Basic, Snobol, entre otros gracias al surgimiento de los compiladores e interpretes (se abundará sobre este particular más adelante).

Durante esta etapa la fabricación de computadoras ingresa a una etapa industrial dónde muchas empresas pueden tener acceso a estas tecnologías a pesar de no ser aún masivas.

2.1.4.3. Paradigma Estructurado

El nacimiento del paradigma Estructurado se basa en el teorema de la estructura (ya comentado) de Böhm y Jacopini de 1966 y la publicación de los trabajos de Dijkstra y Hoare, ratificados después por Harla D. Mills [3] seguidos posteriormente por Yourdon y Constantine [105] así como de DeMarco [34] y otros autores.

Alcalde [3] señala que la Programación Estructurada utiliza todas las estructuras de datos, tanto simples como complejas, que se han desarrollado hasta el momento y que propone tres conceptos fundamentales: Una metodología formal de diseño basada en la descomposición funcional descendente (top-down). Un conjunto de recursos abstractos basados en la modularidad y las características de la misma (cohesión y acoplamiento) y un conjunto de estructuras básicas de programación.

El proceso de descomposición funcional se basa en el principio básico de la división del problema en subtarear y las subtarear en subsubtarear y así sucesivamente hasta identificar tareas concretas y específicas. Este proceso repetitivo de descomposición funcional se le llama refinamientos sucesivos o simplemente *refinamiento*. El modelado resultante de la descomposición funcional es un diagrama muy similar al de un organigrama. Como el proceso se realiza de tareas abstractas (o también llamadas de alto nivel) hasta tareas concretas (también llamadas de bajo nivel) se dice que el proceso se hace de arriba hacia abajo (top-down en inglés) aunque también existe la posibilidad de hacerlo de abajo hacia arriba (bottom-up) si resulta conveniente para el proyecto.

El concepto de abstracción supone, en cada descomposición que todas las partes identificadas están resueltas, dejando su realización para el siguiente refinamiento y considerando que todas

ellas pueden llegar a estar definidas en instrucciones y estructuras disponibles en los lenguajes de programación.

Los módulos resultantes del proceso de refinamiento tienen dos características: cohesión y acoplamiento. La cohesión se refiere al grado de independencia de un módulo para con los otros; es decir, se busca que un módulo pueda realizar una tarea por si mismo sin necesidad de utilizar rutinas de otros módulos. En la medida en que el módulo puede resolver la tarea por si mismo se dice si tiene alta o baja cohesión. La alta cohesión es algo normalmente deseable. El acoplamiento es la medida de interconexión que tiene un módulo en particular con otros. Mientras menos comunicación exista entre un módulo con otro se dice que hay bajo acoplamiento. El bajo acoplamiento es algo normalmente deseable.

El conjunto de estructuras básicas se divide en estructuras de datos y en estructuras de control. Las de datos clasifican a todas aquellas que hasta el momento han sido creadas (mencionadas en párrafos anteriores). Las de control se basan únicamente en tres tipos de estructuras primitivas: Secuencia, repetición y condición (ver figura 2) descartando totalmente las de salto condicional y salto in condicional del paradigma anterior (aunque algunos lenguajes por compatibilidad hacia atrás conservaron mecanismos de salto).

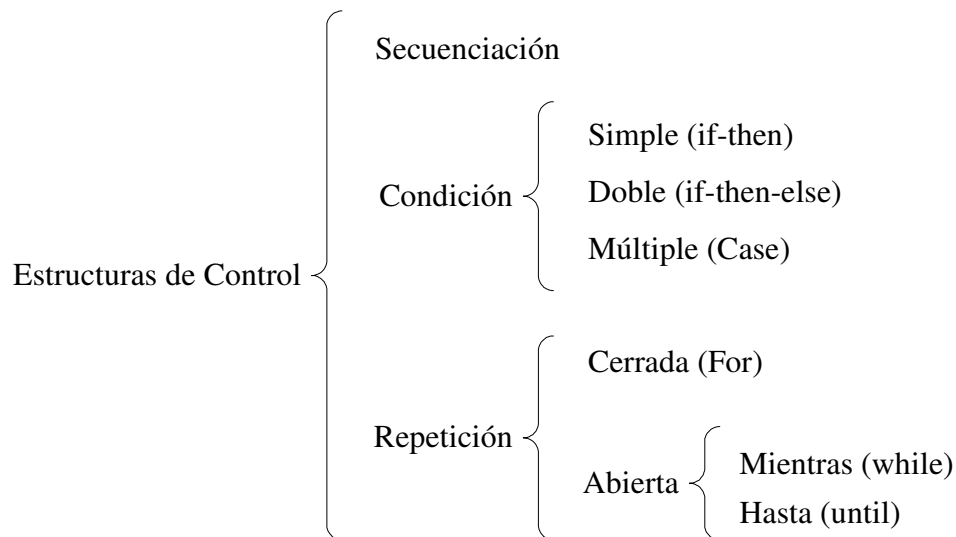


Figura 2. Estructuras de control para el paradigma estructurado

Las estructuras de repetición se dividen en dos categorías: cerradas y abiertas. Las de repetición cerradas son aquellas que señalan de antemano el número de repeticiones que deben hacerse, por ejemplo: repite cinco veces este código; repite n veces este código (teniendo n un valor conocido antes de ejecutar la instrucción). La implementación típica en un lenguaje es la instrucción *for*. Las de repeticiones abiertas son aquellas en las que se desconoce el número de iteraciones. Hay dos tipos: las repeticiones mientras se cumple una condición y las repeticiones hasta que se cumpla una condición. Las implementaciones típicas son *until* y *while*.

Las estructuras condicionales se dividen en tres tipos: simples, dobles y múltiples. Las simples son las que ejecutan una sección de código en caso de que una condición booleana se cumpla. Las dobles son aquellas que ejecutan una sección de código al cumplirse la condición booleana y

otra sección diferente al no cumplirse. Las múltiples son aquellas que ejecutan diferentes secciones de código según el valor de una expresión. Las implementaciones típicas en los lenguajes son: *if-then*, *if-then-else* y *case*.

Considerando la secuencia, tres estructuras de repetición y tres de condición se forman siete estructuras de control básicas con las que se puede construir cualquier algoritmo.

2.1.4.4. Paradigma Orientado a Objetos

El surgimiento del Paradigma Orientado a Objetos se da como resultado de una combinación de ideas, entre ellas los ADT, el ocultamiento de información y el acceso uniforme (todas ellas ya comentadas en la sección de teorías). Antes de señalar las características de este paradigma es importante que se haga una reflexión sobre su nacimiento y difusión en los centros educativos.

Uno de los principales obstáculos que ha tenido la Programación Orientada a Objetos es producto de la estrategia de enseñanza que en la década pasada sus propios promotores realizaron al tratar de romper completamente con la programación estructurada e impartir en clase, como primer contacto con la programación, la orientada a objetos.

La gran diferencia entre la programación estructurada y la programación orientada a objetos radica en la filosofía de enfrentar problemas. Mientras que la primera propone una descomposición funcional del problema, la segunda identifica actores y sus responsabilidades (esto incluye identificar casos de uso, clases y mensajes fluyendo entre los objetos). Esta diferencia en la forma de pensar y la dificultad que representaba para los programadores expertos tener un cambio cultural motivó a los educadores a abandonar casi completamente la enseñanza de la programación estructurada en las escuelas con la intención de eliminar el trauma de la transición de un paradigma al otro en los novales estudiantes.

Este razonamiento en principio puede parecer lógico, sin embargo hay una sutileza que fue pasada por alto que ocasionó una crisis en la formación de programadores. Esta sutileza es confundir la ingeniería de software estructurada y la consecuente programación basada en módulos y funciones con el uso de las estructuras de control formales propuestas por Böhm y Jacopini. A pesar de que la teoría de la estructura es el basamento del paradigma estructurado no está de ninguna manera divorciado del enfoque orientado a objetos. De hecho, toda implementación de método alguno, dentro de cualquier clase, utiliza las tres estructuras de control propuestas por esta teoría lo que demuestra su vigencia y necesidad de aprendizaje y dominio.

Este error fue detectado y señalado por importantes autores de la época. Budd [17] al hablar sobre la enseñanza de la programación orientada a objetos señala: “El libro *no* debe considerarse como un sustituto de un curso introductorio o un manual de referencia de cualquiera de los cuatro lenguajes discutidos....

“El material presentado en este libro supone solamente que el lector es conocedor de algún lenguaje de programación convencional, tal como Pascal o C”. Nótese que Budd señala la necesidad del conocimiento de un lenguaje estructurado como prerrequisito. Cox [28], sobre este particular escribe: “Aunque el punto de destino es una revolución del software, este libro insiste

en alcanzar este destino mediante extensiones evolutivas de los lenguajes actuales de programación en lugar de abandonar la compatibilidad con el pasado”. Otro autor, Voss [102], es mucho más explícito en sus comentarios, se citan tres de ellos: “Los programadores acostumbrados a pensar en términos de procedimientos con frecuencia se sienten desorientados al tratar de comprender la finalidad de los objetos y la estructura de los programas OO. Escuchan una y otra vez que la programación orientada a objetos requiere una forma de pensar completamente nueva: olviden todo lo que ya saben. Nada podría estar más alejado de la realidad. Sin importar lo que alguien diga, mientras más práctica tenga usted en los principios de buen diseño y codificación estructurados, le será más fácil la transición a objetos.”. También comparte la opinión de Budd al señalar que todo programador que desee involucrarse con la orientación a objetos “Debe tener familiaridad básica con al menos un lenguaje procedural como C o Pascal”. Finalmente exclama: “Algunos programadores han llegado a creer que los principios de la programación estructurada son obsoletos y que los ha reemplazado la programación Orientada a Objetos. Hoy en día, más que nunca los principios de la programación estructurada se mantienen firmes. La programación orientada a objetos complementa estos principios.”.

Es particularmente notorio como estos autores subrayan la importancia de desarrollar una lógica de programación basada en las estructuras de control básicas como habilidades previas al dominio de la programación orientada a objetos pero las herramientas de modelado actuales (incluyendo el propio UML) no incluyen sintaxis para esa semántica.

Regresando a la esencia de la discusión de este apartado se debe señalar que los principios fundamentales de la programación orientada a objetos son el Encapsulamiento, la Herencia y el Polimorfismo. El estudio detallado de estas características va más allá de los objetivos de este trabajo así que únicamente se citarán definiciones genéricas para su identificación. Meyer [71] define la POO como “La construcción de software orientado a objetos es el método de desarrollo de software que basa la arquitectura de cualquier sistema de software en módulos deducidos de los tipos de objetos que manipula (en lugar de basarse en la función o funciones a las que el sistema está destinado a asegurar).”

El encapsulamiento es la capacidad que tiene un objeto de ocultar la implementación de los métodos a sus usuarios. A través de una interfaz se puede tener acceso a las propiedades (datos internos del objeto) y solicitud de operaciones (invocación de métodos).

La herencia es la capacidad que tiene una subclase (o clase derivada) de reutilizar el código de una superclase (o clase original). Esto incluye tanto sus métodos como sus propiedades. Este código puede ser refinado, derogado o utilizado tal cual se encuentra en la superclase.

El polimorfismo es la capacidad que tiene un objeto de comportarse de distintas formas a los mensajes que recibe de otros objetos. Esta capacidad puede ser el resultado de la sobrecarga, de la coerción (casting) o de otras técnicas similares.

2.1.4.5. Otros Paradigmas

En los últimos años han nacido nuevos principios que aspiran a convertirse en paradigmas. Entre ellos los más sobresalientes son el cómputo distribuido a través de componentes y la

programación orientada a aspectos (ambos son conceptos evolucionados de los objetos). Para los fines de esta investigación, las aportaciones nuevas que estos paradigmas han realizado al diseño algorítmico son marginales.

2.1.5. Evolución de los Lenguajes de Programación

Hablar de la evolución de los lenguajes de programación es sinónimo de hablar de la historia misma de la computación y aunque pareciera innecesario abordarlo será tocado bajo la perspectiva de su vinculación con la teoría misma de la programación y rescatar los procesos de diseño algorítmicos existentes.

Un *Lenguaje de Programación* según Alcalde [3] es “una notación para escribir programas, es decir, para describir algoritmos dirigidos a la computadora.

“Un lenguaje viene dado por una gramática o conjunto de reglas que se aplican a un alfabeto.”. Por su parte, Glenn [45] expresa su opinión de la siguiente forma: “Una primitiva consiste en una estructura semántica bien definida junto con una sintaxis no ambigua para representarla. ... La colección de primitivas, junto con las reglas de combinación para representar estructuras complejas, constituyen un lenguaje de programación”.

Partiendo de estas definiciones se observa que todo algoritmo será finalmente codificado en un lenguaje de programación y todo lenguaje de programación tiene por objetivo ser una notación para que un algoritmo pueda ser ejecutado por una computadora. Por tanto, las técnicas de modelado de algoritmos están estrechamente vinculadas con la semántica de los lenguajes de programación, y está, a su vez con los paradigmas dictados por la teoría de la programación.

2.1.5.1. De la Programación Dura al Macroensamblador

Como ya se citó anteriormente en la época de la programación dura en realidad no había técnicas de modelado algorítmico y la forma de programación era a través del alambrado físico del equipo para realizar una tarea específica. Una vez alambrada la computadora se alimentaban los datos y estos eran procesados por el equipo según el alambrado vigente.

Posterior al surgimiento del modelo de Neumann aparecen los primeros lenguajes de programación los cuales en sus inicios se escribían directamente en código máquina. Esto significa codificar directamente en binario. Muy poco después, con la intención de facilitar esta programación, se codificaron los números binarios en hexadecimal generando piezas de código como la siguiente:

```
1E B80000 50 B82810 8ED8 8EC8 BF0000 BB1D00 8B0F BB1F00 8A07 FC F2 AE 7401  
CB 4F CB
```

Este es un ejemplo real escrito por Levine [65] en código del microprocesador Intel 8086 que sirve para encontrar un número entre un conjunto de enteros mediante el método de búsqueda lineal. Cada grupo de número representa una instrucción para el microprocesador. Dentro de sus características está el hecho de que hacen uso de referencias a celdas absolutas en la memoria y

es difícil de cambiarlo o adaptarlo a nuevos requerimientos debido a que son un tanto inteligibles. No hay, en este período técnicas especiales de diseño algorítmico ni de depuración, solamente prueba y error basado en el talento del programador.

El siguiente paso fue el lenguaje ensamblador. El ensamblador es un traductor de palabras simples llamadas mnemónicos (algunos autores los llaman mnemotécnicos) a las instrucciones en hexadecimal (o binario). El resultado fue el nacimiento de un lenguaje en el que cada instrucción binaria del microprocesador tenía un mnemónico equivalente. Los programas siguieron siendo extensos y difíciles de mantener o modificar pero dieron un primer gran paso en la legibilidad ya que ahora se podían observar palabras como ADD, STR, JMP, etc. Para efectos del diseño algorítmico la única posible aportación es que pronto se descubrieron secuencias de instrucciones que se repetían para realizar alguna tarea en particular, fuera de eso no hay aportaciones importantes en este momento.

El hecho de descubrir secuencias de mnemónicos repetitivos fue el paso para crear el macro ensamblador. Este lenguaje cuenta con palabras reservadas que ya no corresponden a una instrucción específica del procesador, sino a un conjunto de ellas. Cuando un programador utiliza alguna palabra *macro* el macro ensamblador la sustituye por el conjunto de mnemónicos correspondientes. Una vez más se daba otro paso hacia la legibilidad de la programación. Sin embargo, las herramientas formales de diseño algorítmico aún no están desarrolladas.

2.1.5.2. Del Macroensamblador a los Lenguajes de Alto Nivel. Primera Oleada

Gracias al éxito de los macro ensambladores y al desarrollo de la teoría de lenguajes formales pronto se pensaba en la elaboración de un lenguaje de *alto nivel* entendiendo este concepto como aquellos lenguajes cuya legibilidad y capacidad de expresión sean tales que un ser humano pueda leer y comprender rápidamente el contexto de su misión.

Los avances en los conceptos de análisis lexicográfico, sintáctico y semántico permitieron el nacimiento de los primeros compiladores e interpretes. El primero de un nuevo linaje de lenguajes de programación denominados 3GL (por el concepto de lenguajes de tercera generación) fue Fortran que nació en 1957 seguido rápidamente por toda una constelación de lenguajes como COBOL (1958), LISP (1958), ALGOL (1960), APL (1961), BASIC (1964), PL/1 (1965), Simula (1967), etcétera [65]. Cada uno de ellos con su propia visión sobre como debe hacerse la programación acorde con las teorías de la época.

Esta es una época muy importante en el desarrollo algorítmico. Con el advenimiento masivo de lenguajes de alto nivel aquellos pocos programadores de la época de los lenguajes de bajo nivel se multiplicaron hasta convertirse en ejércitos de personas que participaban en el desarrollo de toda clase de proyectos de software: Sistemas operativos, compiladores, manejadores de bases de datos, aplicaciones administrativas, científicas, militares, etc. Las estructuras de datos formales acompañadas de sus algoritmos hacen aparición en escena. Los primeros esfuerzos en inteligencia artificial surgen con los lenguajes funcionales y lógicos apartándose del modelo imperativo. Más adelante, con la publicación del modelo relacional de Codd [26] comienzan a parecer los lenguajes 4GL (por ser denominados de cuarta generación) basados en el SQL, resultado del álgebra y el cálculo relacional de tuplas.

Con todos estos actores en escena surgen toda una colección de algoritmos, herramientas y técnicas para su diseño y construcción. Herramientas como los diagramas de bloques, diagramas de flujo, HIPO, Pseudocódigo, tablas de verdad, tablas de decisión, árboles de decisión, etc. surgen y se multiplican, en una primer instancia de forma un tanto anárquica pero poco a poco van surgiendo los estándares como por ejemplo [3]: ANSI X3.5-1070; ISO 1028-1973; AFNOR Z 670 10; IRNAOR PNE 71-001, etc.

2.1.5.3. Segunda Oleada

Después de la primera oleada de lenguajes 3GL, todos utilizando en sus inicios las estructuras de la programación libre; viene una segunda oleada de lenguajes que implementan el modelo estructurado. Entre los nuevos lenguajes nacen Pascal (1970), C (1972), Prolog (1972) y muchos de los anteriores hacen cambios en sus estructuras para implementar de lleno las ideas de Böhm y Jacopini. Sin embargo, varios de ellos conservan las estructuras de salto, tanto incondicional como condicional, como mecanismo de compatibilidad con el pasado.

Las herramientas ya existentes se perfeccionan y hacen su aparición técnicas de construcción y modelado de programas como las de Bertini, Jackson, Warnier/Orr, Chapin, Nassi/Shneiderman, Tabourier, DeMarco, etc. (estas técnicas son abordadas con detalle más adelante).

2.1.5.4. Tercera Oleada

Finalmente, durante los 80's hacen su aparición una nueva oleada de lenguajes cuya sintaxis soporta la programación orientada a objetos y se repite el proceso de adecuación de los viejos lenguajes a los nuevos paradigmas. Por supuesto, también nacen nuevos como Modula2 (1979), ADA (1983) y Java (1991) entre otros. Durante esta época existe cierto desconcierto en materia de diseño algorítmico. Las herramientas tradicionales son abandonadas y las técnicas de construcción estructurada de programas también. Nuevos procesos para identificar responsabilidades aparecen como el uso de las tarjetas CRC [17], los diagramas de jerarquías de clase [13] y los diseños por contrato [62], entre otros.

Desde el punto de vista de diseño algorítmico, la herencia genera un gran impacto ya que distribuye las responsabilidades algorítmicas de los métodos en diversas clases que especializan, usan o derogan las acciones. Por otra parte, el encapsulamiento fuerza al diseño algorítmico a ver a los datos y los procesos como una sola cosa y no como entes separados como sucedía en el modelo estructurado. Finalmente el polimorfismo impacta el diseño de algoritmos llevándolo a visualizar comportamientos alternos ante una misma invocación. Tomo tiempo antes de que se estandarizara el modelado de programas a través de un lenguaje común aceptado por la industria y ver el nacimiento de UML [62].

Con estos advenimientos ya el cuadro está casi completo: los lenguajes imperativos apegados totalmente al modelado algorítmico, los funcionales basados en funciones matemáticas y los lógicos basados en la lógica matemática y cálculo de preposiciones. El escenario está dispuesto para la llegada de los nuevos chicos de la cuadra: Los lenguajes descriptivos.

2.1.5.5. Los Lenguajes Descriptivos

Los lenguajes descriptivos tienen por objetivo modelar escenarios y describir como son las cosas en un mundo particular. Entre ellos se encuentra el VRML [21], que es un lenguaje para crear mundos virtuales, y el HTML [64], que se utiliza para crear páginas Web, entre otros. Debido a que su objetivo computacional es diferente y las técnicas de diseño propias de su modelo no implican el modelado algorítmico. El nacimiento de este tipo de lenguajes no aporta nuevos elementos para el tema de estudio de este trabajo ya que su uso no requiere el diseño algorítmico.

2.1.5.6. Taxonomía de la Programación

Con la información que se ha obtenido es posible construir una taxonomía de la programación basada en el tipo de programación. Debido a que los paradigmas dictan normas de conducta y filosóficas de cómo enfrentar el problema no serán utilizados en este ejercicio.

Hasta el momento se han detectado cuatro modelos de programación básica: imperativa, funcional, lógica y descriptiva. Mismos que ya han sido comentados y que se presentan en la figura 3 con ejemplos de sus lenguajes asociados.

A pesar de que cada uno de estos modelos de programación tiene su riqueza y nichos específicos de uso y operación para los fines de este trabajo el modelo imperativo está mucho más vinculado con el modelado algorítmico convencional y es, por lo tanto, la frontera de estudio. Por ello mismo se hace una pequeña ampliación de su taxonomía, más con un interés enunciativo que de ser exhaustivo.

2.1.6. El Linaje de la Programación

En la figura 5 se muestra un cuadro del linaje de los lenguajes de programación clasificados a la luz de la taxonomía, paradigma y relación entre ellos según la información que se ha expuesto. En ella se puede observar la evolución de los lenguajes y como algunos toman cosas de lenguajes previos creándose una secuencia de ancestros y descendientes.

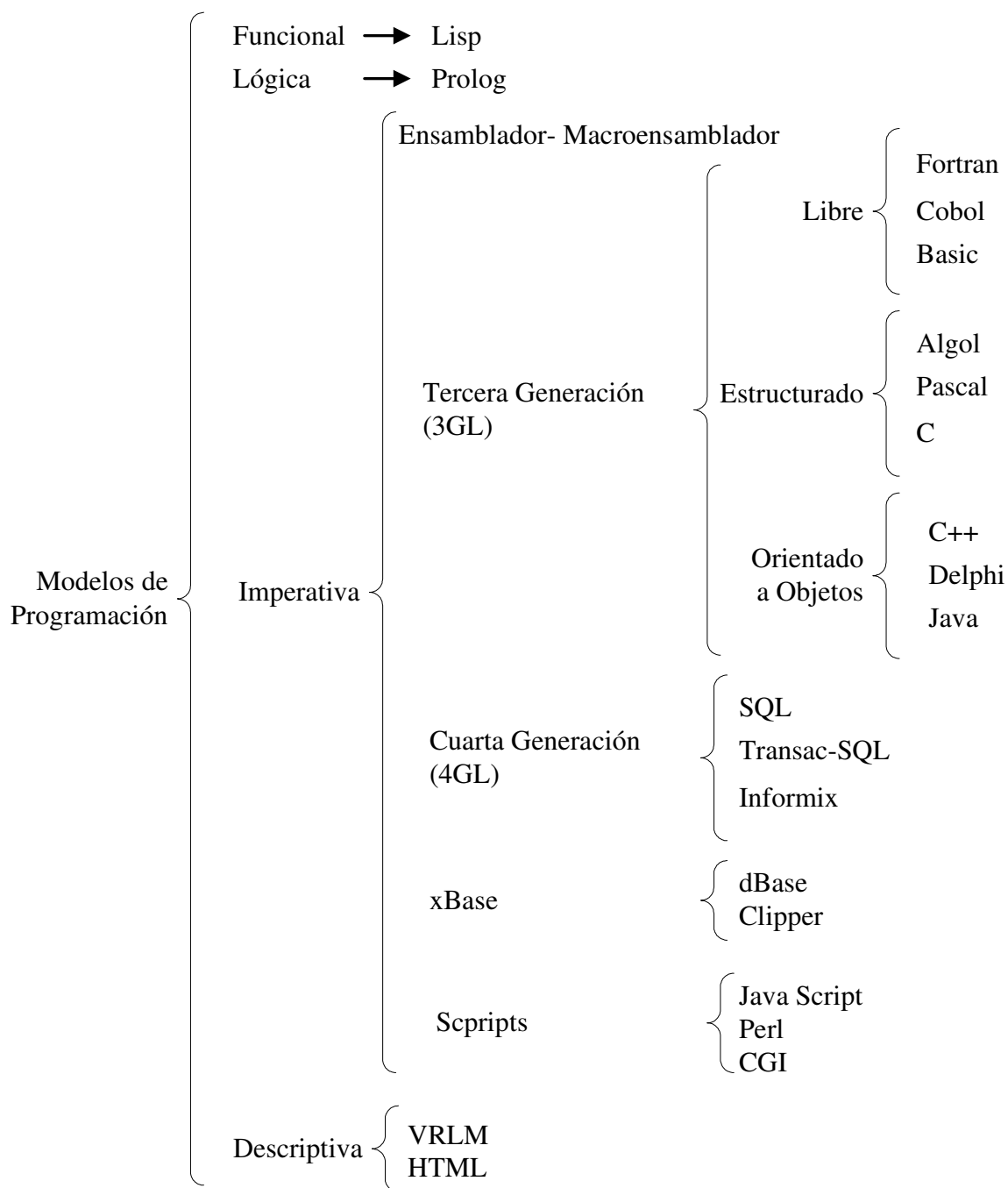


Figura 3. Taxonomía de los modelos de programación con ejemplos

2.1.7. Mapa Mental de la Lógica de Programación

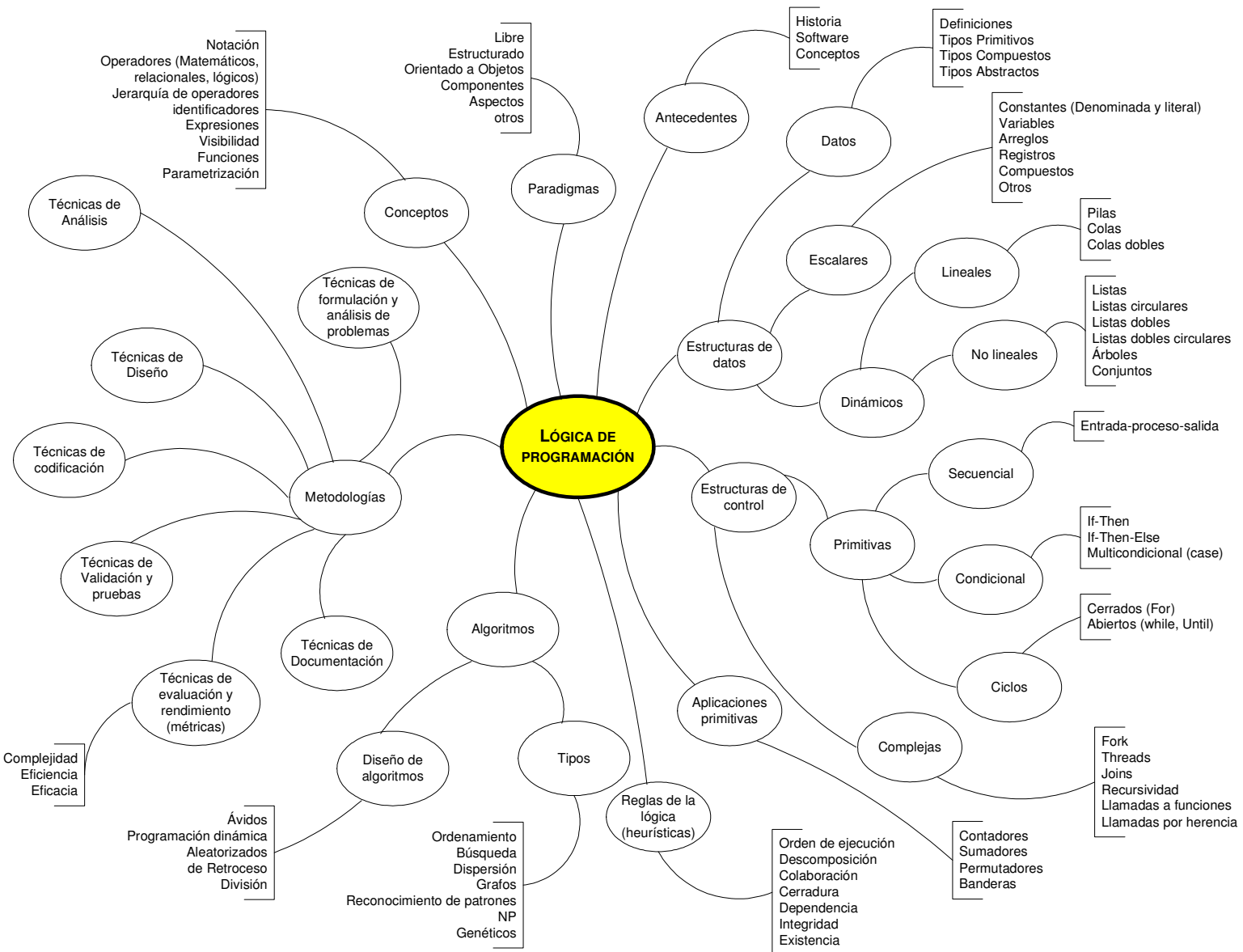


Figura 4. Lógica de programación

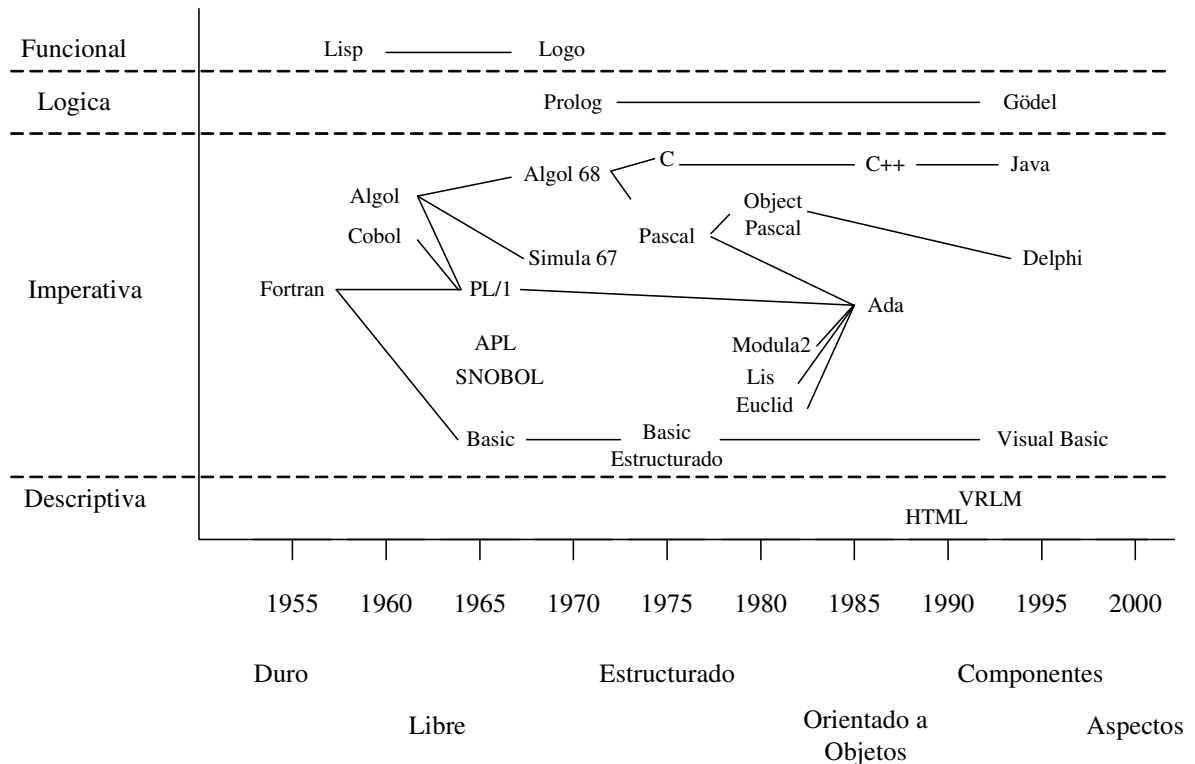


Figura 5. Linaje de los lenguajes de programación

2.2 Diseño de Algoritmos

El diseño de algoritmos es la parte central de todo producto de software sin importar el paradigma en que se esté desarrollando. Si se trata de programación estructurada los módulos, rutinas y funciones subyacen en algoritmos. Si se trata de objetos, cada método de cada clase contiene un algoritmo. Si se trata de cómputo colaborativo y distribuido, cada componente disperso se basa en algoritmos. A pesar del paso de los años el diseño de algoritmos imperativos se basa en la correcta aplicación del teorema de Jacopini del cual se deriva el concepto de “lógica de programación” de Vargas [101].

2.2.1. Conceptos

Un algoritmo según Johnsonbaugh [54] es un conjunto finito de instrucciones que tiene las siguientes características:

- ✓ Precisión: todos los pasos son descritos con exactitud y sin ambigüedad
- ✓ Unicidad: los resultados intermedios de la ejecución de cada paso son únicamente definidos y dependientes de los resultados y las entradas de los pasos anteriores.
- ✓ Finito: el algoritmo se detiene después de haber sido ejecutadas varias instrucciones.
- ✓ Entrada: el algoritmo recibe entradas.
- ✓ Salida: el algoritmo produce salidas.
- ✓ Generalidad: el algoritmo aplica a un conjunto de entradas.

Prácticamente todos los libros que abordan el tema de algoritmos se concentran en las definiciones, la complejidad, las métricas y una descripción de los algoritmos típicos desde los simples hasta los complejos pasando por búsquedas, dispersión, clasificación y los algoritmos que rigen el comportamiento de las estructuras de datos dinámicas como las pilas, colas, listas, árboles, etc. Sin embargo, la temática que nos ocupa se concentra en la forma como las personas diseñan algoritmos, las herramientas de diseño que se han utilizado, las técnicas de aproximación y los procesos mentales involucrados durante el diseño. Las métricas y análisis de complejidad se realizan cuando ya se cuenta con un algoritmo, no antes.

Ya que el interés principal es como encontrar algoritmos, en las siguientes secciones se aborda cómo, a lo largo de los años, se ha enfrentado el complejo proceso de creación de algoritmos.

2.2.2. Métodos de Análisis de Problemas

Todo software, sin importar su naturaleza, en esencia lo que busca es resolver un problema de la vida real automatizando una serie de actividades. Esta automatización está codificada en un lenguaje de programación que, como ya se explicó con anterioridad, no es más que una sintaxis para expresar un algoritmo de forma que una computadora pueda ejecutarlo. Diseñar el algoritmo y codificarlo es de hecho encontrar la solución al problema. Glenn [45] lo explica de esta forma: “Descubrir un algoritmo para resolver un problema es equivalente, en lo esencial a descubrir una solución para ese problema”.

Todo el esfuerzo e inteligencia requeridos para encontrar la solución a un problema se encuentran de alguna forma, *encapsulados* dentro del algoritmo. El mismo Glenn [45] haciendo una interesante reflexión afirma lo siguiente: “Una vez descubierto un algoritmo para efectuar una tarea, la realización de esta ya no requiere entender los principios en los que se basa dicho algoritmo, pues el proceso se reduce a seguir ciertas instrucciones. En cierto sentido, la inteligencia requerida para llevar a cabo la tarea está codificada en el algoritmo. Es gracias a esta capacidad para capturar y comunicar inteligencia mediante algoritmos que se pueden construir máquinas cuyo comportamiento simula inteligencia. En consecuencia, el nivel de inteligencia que exhiba una máquina estará limitado por la inteligencia que se pueda comunicarle por medio de un algoritmo”.

Es la inteligencia humana la que permite encapsular *pedazos* de inteligencia dentro de programas a través de los algoritmos. Sin embargo, este proceso no es de ninguna manera trivial. Encontrar soluciones a problemas es en sí mismo un problema. Decker [33] lo ha definido de la siguiente forma: “El mayor obstáculo que enfrentan los novatos al aprender a programar es lo que podríamos llamar ‘El problema de enfrentar una pantalla en blanco’, el cual consiste en como ir de un vago planteamiento de un problema a un programa funcional que lo resuelva”.

Diversos factores intervienen en este proceso de ir de un vago planteamiento a una solución computacional. Entre ellos se encuentran la capacidad de abstracción y el análisis. Joyanes [55] ha agregado la independencia del algoritmo al señalar lo siguiente: “El proceso de realización de diferentes pasos hasta encontrar la solución de un problema es un proceso abstracto. Diseñar o concebir un problema en términos abstractos consiste en no tener en cuenta la máquina que lo va

ha resolver así como el lenguaje de programación que se va a utilizar. Esto lleva consigo la obtención de un conjunto de acciones que se han de realizar para obtener la solución”.

Finalmente, no se puede olvidar el factor humano por sí mismo en materia de talento e inspiración. A Edison se le atribuye la frase siguiente: “Mi éxito depende del 98% de sudoración por un 2% de inspiración”. En el área computacional Glenn [45] ha escrito sobre lo que llama irregularidades en el proceso de creación de algoritmos lo siguiente: “Otra irregularidad es la misteriosa inspiración que puede llegar a un potencial solucionador de problemas, quien, después de trabajar con un problema sin éxito aparente, puede ver de repente la solución mientras realiza otra tarea. Helmholtz identificó este fenómeno ya desde 1896, y el matemático Henri Poincaré lo analizó en una conferencia ante la sociedad psicológica de París.... Hoy día, al período entre el trabajo consciente con un problema y la llegada de la inspiración repentina se le llama período de incubación, y su comprensión es una meta de varias investigaciones actuales.”.

Aquí el camino se bifurca. Hay autores que abordan la solución de problemas a través de métodos para elaborar programas (lo cual en estricto sentido es válido si se visualiza el programa como la solución al problema) mientras que otros se van a enfoques más globales al abordar una metodología genérica de solución de problemas. Primero se abordará ésta última.

La resolución de problemas no es una necesidad monopolizada por la computación. Muchas otras ciencias han requerido de esta habilidad desde muchos años antes al nacimiento de la primera computadora. Sin embargo, el método de resolución de problemas más citado es el del matemático Polya. Glenn [45] lo explica adecuadamente de la siguiente forma: “Las fases definidas a grandes rasgos por el matemático G. Polya a finales de los años 40's siguen siendo los principios fundamentales en que se basan los intentos actuales de enseñar a resolver problemas. En realidad los trabajos de Polya se apoyaron en las investigaciones de H. Von Helmholtz [fines del siglo antepasado], J. Dewey [década de los 30's y otros que estudiaron el proceso de resolución de problemas en situaciones generales. Los científicos de la computación suelen citar a Polya porque sus trabajos abordaron la resolución de problemas en el contexto de las matemáticas, pariente cercano de la computación. Las fases de Polya para resolver problemas son:

- Fase 1. Entender el problema.
- Fase 2. Idear un plan para resolver el problema.
- Fase 3. Llevar a cabo el plan.
- Fase 4. Evaluar la solución en cuanto a su exactitud y a su potencial como herramienta para resolver otros problemas.”

El mismo Glenn describe estas fases pero ahora con ojos computacionales llegando a lo siguiente:

- Fase 1. Entender el Problema.
- Fase 2. Pensar como un procedimiento algorítmico podría resolver el problema.
- Fase 3. Formular el algoritmo y representarlo en forma de programa.
- Fase 4. Evaluar el programa en cuanto a su exactitud y potencial como herramienta para resolver otros problemas.

Otro autor, Gosling [47] hace el siguiente resumen: “Esencialmente la técnica consiste en dar una secuencia de sucesos para resolver el problema....Primero comprender el problema...Segundo, idear algún método para obtener una solución....Escribir la solución en términos generales... Detallarla en procesos sucesivos”

Es importante no olvidar que no son pasos sino fases, por lo tanto no tienen que ser seguidas en orden ni tampoco debe haberse concluido una de ellas totalmente antes de iniciar la siguiente. Muchas veces ni siquiera se puede tener un entendimiento del problema si no se cuenta con la solución (lo cual resulta paradójico). Es válido hacer procesos cíclicos comprobando hipótesis para ir logrando la comprensión del problema, por lo tanto no es necesario terminar una etapa antes de iniciar la siguiente (de hacerlo así quizás nunca se encuentre la solución ya que quizás nunca se pueda entender completamente el problema si no se avanza sobre él).

El planteamiento fundamental consiste en dar el primer paso tratando de hacer algún análisis sobre la información que se tiene. Lo que sigue es conservar el impulso del primer paso. Para ello tanto Polya y sus seguidores han identificado al menos tres estrategias:

1. Resolver el problema al revés.- Esto es identificar la salida, luego los procedimientos para obtenerlos, las restricciones y las entradas.
2. Buscar problemas relacionados.- Investigar sobre problemas similares que ya hayan sido resueltos, analizar la solución y tratar de copiarla adaptándola al contexto vigente.
3. Refinamiento por pasos. Dividir el problema en partes (o módulos) y seguir dividiéndolo hasta sus componentes esenciales. Esto puede hacerse con el enfoque top-down o el bottom-up (incluso en el mundo orientado a objetos).

A pesar de que ésta última estrategia de refinamiento y desglose descendente tiene ya varios años de haberse publicado, diversos autores, ya entrados en los 90's, señalaban su vigencia y utilidad. Entre ellos Alcalde [3] en 1992 escribía sobre los refinamientos: “Los programas se diseñan de lo general a lo particular por medio de sucesivos refinamientos o descomposiciones que nos van acercando a las instrucciones finales del programa”. En el mismo libro anota en otra sección [3]: “La programación estructurada y modular insiste en la descomposición en módulos que dividan el problema en sus funciones esenciales (descomposición funcional) y que posteriormente, a través de varios ciclos de refinamiento se obtengan los algoritmos específicos. El proceso de refinamiento cíclico sigue vigente.”. Por su parte, Perry, en 1999 escribe [82] sobre el diseño descendente “El diseño descendente es el proceso de descomposición de un problema en detalles más específicos, hasta completar todos los detalles.”.

Regresando al punto en que los caminos se bifurcaban ahora se abordará el enfoque de elaboración de programas.

Entre las metodologías propuestas que se concentran en la solución de problemas a través de las etapas de elaboración de programas están las de Gosling, Joyanes, Decker, Alcalde, Glenn, Levine y Perry. Entre ellas hay una extrema similitud y retoman ideas de Polya sin hacer mención explícita de su modelo. Se describen a continuación:

Gosling [47] escribe en 1985: “El proceso de diseñar un programa consta de varias etapas. La primera de ellas consiste en la comprensión del problema que se va a resolver mediante un

programa. La segunda en encontrar un método de resolución del problema. A continuación escribimos una solución no en la forma de un diagrama de flujo, sino en la forma de una serie de pasos bien definidos. Estos pasos se establecen en términos amplios en las primeras etapas del desarrollo del programa y luego se van refinando hasta que se obtiene un programa completamente lógico.”.

Joyanes [55] en 1987 explica seis etapas para elaborar un programa de la siguiente forma: “El desarrollo de un programa abarca diferentes etapas, de las cuales la escritura puede ser la menos significativa. La primera es la definición del problema; la segunda es el diseño del algoritmo (algoritmo normalizado); la tercera es la codificación; la cuarta es la depuración; la quinta es la documentación; y la sexta es el mantenimiento.” Siendo el diseño del algoritmo la parte central del proceso describe en [55] las etapas para su desarrollo explicando: “El algoritmo de resolución consta de dos etapas: 1. Diseño del modelo de resolución del problema; y 2. Algoritmo de resolución del problema.”. Finalmente, señala que para poder realizar el diseño de un programa funcionalmente exitoso se deben seguir las siguientes reglas [55]:

1. Ir de lo general a lo particular descendiendo en la estructura del programa hasta su nivel de detalle.
2. De la definición inicial del problema se pasa a un esquema de algoritmo descrito en pseudo código.
3. Independencia inicial del lenguaje.
4. Diseño por niveles, dejando los detalles para niveles posteriores.
5. Finalizar con una recomposición del algoritmo completo.

Levine [66] en 1989 propone un proceso para pasar de la descripción de un problema a un programa definiendo los siguientes pasos:

1. Se propone una solución global al problema en términos de una descripción en un lenguaje llamado pseudo código. Esto será el primer acercamiento a la solución. Este pseudo código describirá de manera aproximada, el procedimiento para resolver el problema.
2. Se toma el módulo recién generado y se comienza a refinar progresivamente, tratando de traducir cada una de sus pseudo instrucciones en órdenes inteligibles para la computadora verificando, a la vez, que esté correcto.
3. Se ejecuta el paso anterior sobre cada uno de los módulos obtenidos, hasta que no quede nada escrito en pseudo código y todo haya sido traducido a lenguaje de computadora”

Más adelante, el mismo autor en [66] propone las siguientes fases para la creación de un programa:

0. Entender el problema.
 1. Hacer el análisis del mismo
 2. Programar el modelo de solución propuesto
 3. Codificarlo
 4. Cargarlo a la computadora para su ejecución y ajuste
 5. Darle mantenimiento durante su tiempo de vida.

Decker [33] en 1992 propone su metodología de elaboración de programas con los siguientes pasos:

1. Diseñar y especificar el programa. Describiendo el problema cuidadosamente, especificando las entradas y salidas así como la interfaz del usuario.
2. Desarrollar el programa. Refinar sucesivamente el esquema original.
3. Analizar el diseño del programa y escribir el código (utilizando técnicas de verificación dónde sea apropiado).
4. Probar el programa y corregir cualquier error detectado lo más detalladamente posible.
5. Mantener, extender y modificar el programa.

Alcalde [3], en 1992 presentó una metodología simplificada en tres fases:

1. Fase de Análisis: consiste en el examen y descripción detallada de la especificación del problema.
2. Fase de Programación: consiste en el diseño de la solución del problema planteado en forma de algoritmo.
3. Fase de codificación: transcripción del algoritmo a un lenguaje de programación.

Glenn [45] en 1995 describe de la siguiente forma las actividades necesarias para la creación de un programa: “La creación de un programa consta de dos actividades: Descubrir el algoritmo básico y representar ese algoritmo en forma de programa. ...el descubrimiento de algoritmos suele ser el paso más difícil en el proceso de creación de software; después de todo, descubrir un algoritmo es equivalente a encontrar un método para resolver el problema cuya solución el algoritmo va a calcular.”

Perry [82] en 1999 propuso una metodología de diseño, acompañada de una serie de pasos para hacer el diseño descendente, que se describe a continuación:

Metodología de diseño:

1. Defina la salida.
2. Desarrolle la lógica para lograr esta salida.
3. Escriba el programa.

Pasos para el diseño descendente:

1. Determinar la meta general.
2. Dividir esa meta en dos o tres partes más detalladas.
3. Posponer los detalles mientras sea posible. Repetir los pasos 1 y 2 hasta que ya no pueda dividir más el problema.

Es importante señalar que los autores citados tienen la particularidad de que sus ideas están enfocadas en los microprocesos de elaboración de programas y definición de algoritmos no proponen sus metodologías para enfrentar el análisis y diseño de sistemas ni como técnicas de ingeniería de software. Para ello Senn [91], Kendal [58], McMillan [70], Pressman [86], Sommerville [97] y otros han expuesto metodologías formales.

2.2.3. Métodos de Diseño de Algoritmos

El auge de los métodos de diseño de algoritmos se da en el seno del paradigma estructurado dónde resaltan las siguientes: Warnier/Orr [103], Jackson [53], Bertini [3] [55], Tabourier [3] y HIPO [91] aunque este último en realidad surgió bajo el paradigma libre. Todos ellos plantean el hecho de que un algoritmo no se obtiene al primer intento sino que deben de hacerse varias aproximaciones a través de mecanismos de refinamiento. Todos utilizan algún tipo de diagrama como herramienta de diseño.

Se abundará en cada uno de estos métodos identificando sus rasgos principales sin aspirar a ser una manual detallado ellos.

Los diagramas de Warnier/Orr, explica Senn [91], (también conocidos como construcción lógica de programas/construcción lógica de sistemas) fueron desarrollados inicialmente en Francia por Jean-Dominique Warnier y en los Estados Unidos por Kenneth Orr. Joyanes [55] explica este método diciendo: “Se basa en una metodología matemática que establece un único lenguaje entre usuarios, analistas y programadores.

“El método se basa en la descomposición por niveles del problema. En cada nivel se detallan los tratamientos que permiten la solución del problema planteado.

Se representa con llaves y las estructuras son conceptualmente idénticas a las de Jackson.”

Alcalde [3] es quizás el que los desarrolla con más detalle describiendo la metodología, sus fases, los diagramas y los tipos de instrucciones. La metodología se basa fundamentalmente en la jerarquía de los datos, tanto de entrada como de salida, siendo estos últimos los que de forma directa ejercen una influencia esencial en la definición y control del programa.

Esta metodología utiliza varias herramientas: (Archivos lógicos, tablas de verdad y diagramas de Veicht) y un conjunto de normas.

Representa a las estructuras de dos formas diferentes:

- Cuadro de Descomposición de Secuencias. Representación basada en el uso de llaves dónde desde el principio (parte superior) hasta el fin (parte inferior) aparecen los elementos integrantes de la estructura y su número de ocurrencias.
- Organigrama de secuencias. Organigrama que utiliza los símbolos normalizados con un estilo propio. Empieza con un principio y termina con un fin.

El elemento esencial en el que se basa el método es la SECUENCIA LÓGICA compuesta por un conjunto de sentencias o estructuras que se ejecutan el mismo número de veces y en el mismo orden.

Las fases que sigue la metodología se explican en [3] y son las siguientes:

1. Estudio de los datos de salida. Trata de crear el archivo lógico de salida (ALS).
2. Estudio de los datos de entrada. Trata de crear el archivo lógico de entrada (ALE), teniendo en cuenta la organización de los datos de salida y los posibles datos intermedios.

3. Hacer el cuadro de descomposición de secuencias.
4. Dibujar el organigrama de secuencias de Warnier
5. Construir la lista de instrucciones y asignarlas en el organigrama de secuencias.
6. Diseñar el juego de datos de ensayo y analizar los resultados.

Ente las fases 2 y 3 debe hacerse el estudio de los datos intermedios necesarios pero que no pertenecen a la entrada ni a la salida.

El diagrama de Warnier hace empleo de llaves de distintos tamaños que relacionan entre si todas las tareas y operaciones.

La representación del algoritmo se basa en los siguientes puntos:

- Un programa se representa por un único diagrama en el cual se engloban todas las operaciones necesarias para la resolución del problema. Estas operaciones están colocadas secuencialmente a la derecha de una llave, en cuya parte izquierda figura el nombre del programa.
- En la parte superior de la llave anterior figurará el comentario Inicio.
- En la parte inferior figurará Fin.
- La forma de conectar con distintas páginas es a través de la palabra proceso seguida de un número o un nombre que tenga relación con las operaciones que se realizan en la siguiente página. Estas palabras figurarán en el diagrama principal (diagrama que ocupa la primera página). En las siguientes figurará un diagrama sujeto a las mismas normas, salvo que el nombre del programa será la palabra anteriormente citada. Las sucesivas conexiones se hacen en forma similar.
- Las estructuras tienen dos formas de representación.

Las instrucciones se agrupan en dos secciones:

- Instrucciones de tratamiento. Son las instrucciones primitivas cuyo cometido es la realizar una operación de forma inmediata. (Lectura de un valor, o un determinado cálculo).
- Instrucciones de estructura. Son las instrucciones de control de programa. Alternativas (condiciones) y repeticiones.

En la figura 6 se muestra la sintaxis de los diagramas de Warnier/Orr para representar cada una de las estructuras de control del teorema de Jacopini.

En la figura 7 se muestra el diseño completo de un algoritmo que realiza la lectura de tres números y los imprime ordenados ascendentemente señalando si los números fueron ingresados en orden o no. Este proceso lo repite cinco veces.

El método de Jackson, explica Joyanes en [55] fue creado por el Inglés Michael Jackson se basa en que la estructura de un programa está en función de la estructura de los datos que manipula. Emplea módulos según un orden jerárquico dentro de los diferentes niveles dónde se encuentra.

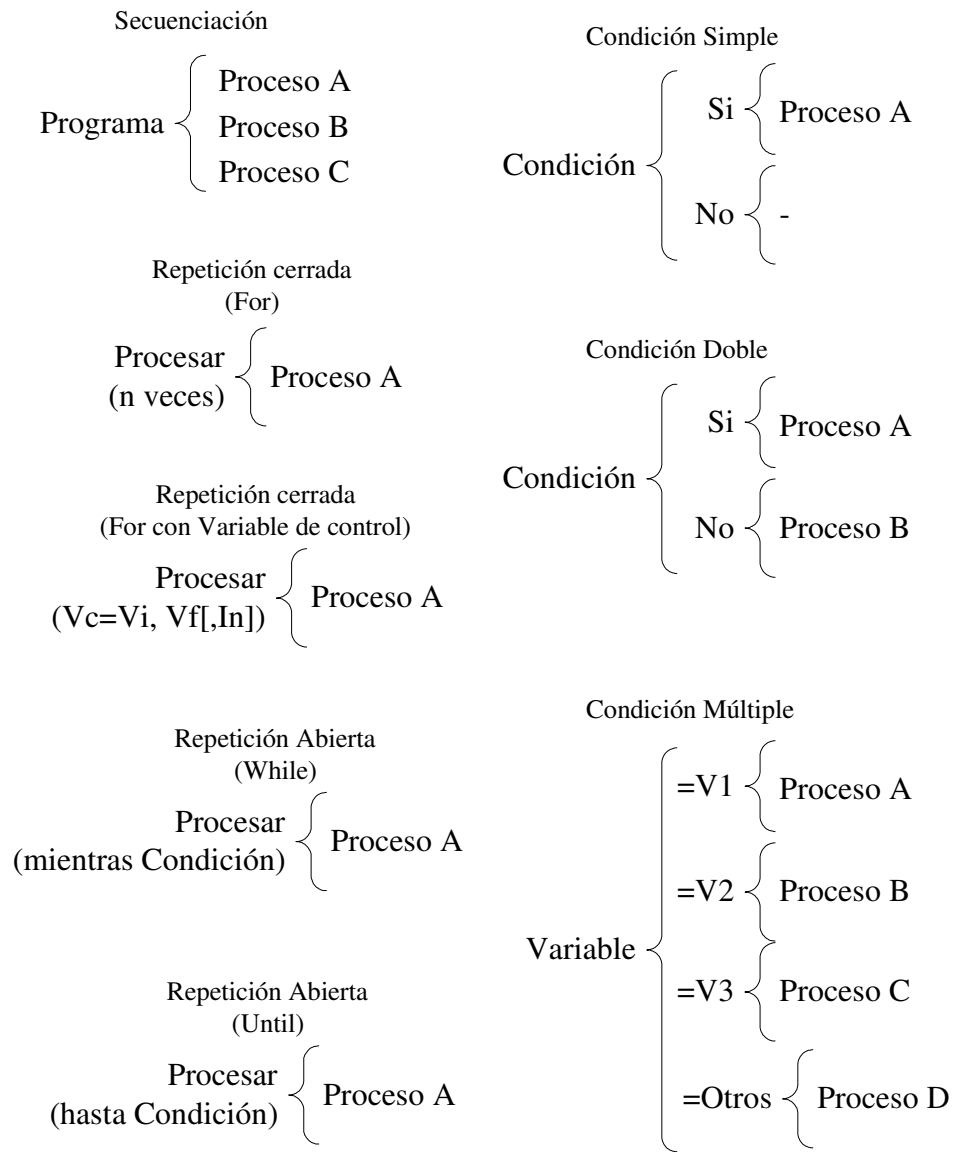


Figura 6. Sintaxis de los diagramas de Warnier/Orr

La metodología es explicada por Alcalde en [3] de la siguiente forma: “El punto de partida es el de la definición de los datos que vamos a manejar, tanto en la entrada como en la salida, y como consecuencia del tratamiento requerido en estos datos se obtiene el programa.

“Como podemos observar, la determinación de las FUNCIONES y los DATOS a manejar se obtienen de las especificaciones y requisitos del usuario. Las ESTRUCTURAS DE DATOS (tanto de entrada como de salida) se forman a partir de los datos. Mediante determinación de RELACIONES y CORRESPONDENCIA entre las distintas estructuras de datos que intervengan se obtiene la ESTRUCTURA DEL PROGRAMA. De las funciones a realizar se deducen las OPERACIONES que serán necesarias en el programa. A la estructura del programa se le ASIGNAN las operaciones necesarias u el resultado se escribe en una notación propia similar a un pseudo código denominado LÓGICA ESQUEMATIZADA.”

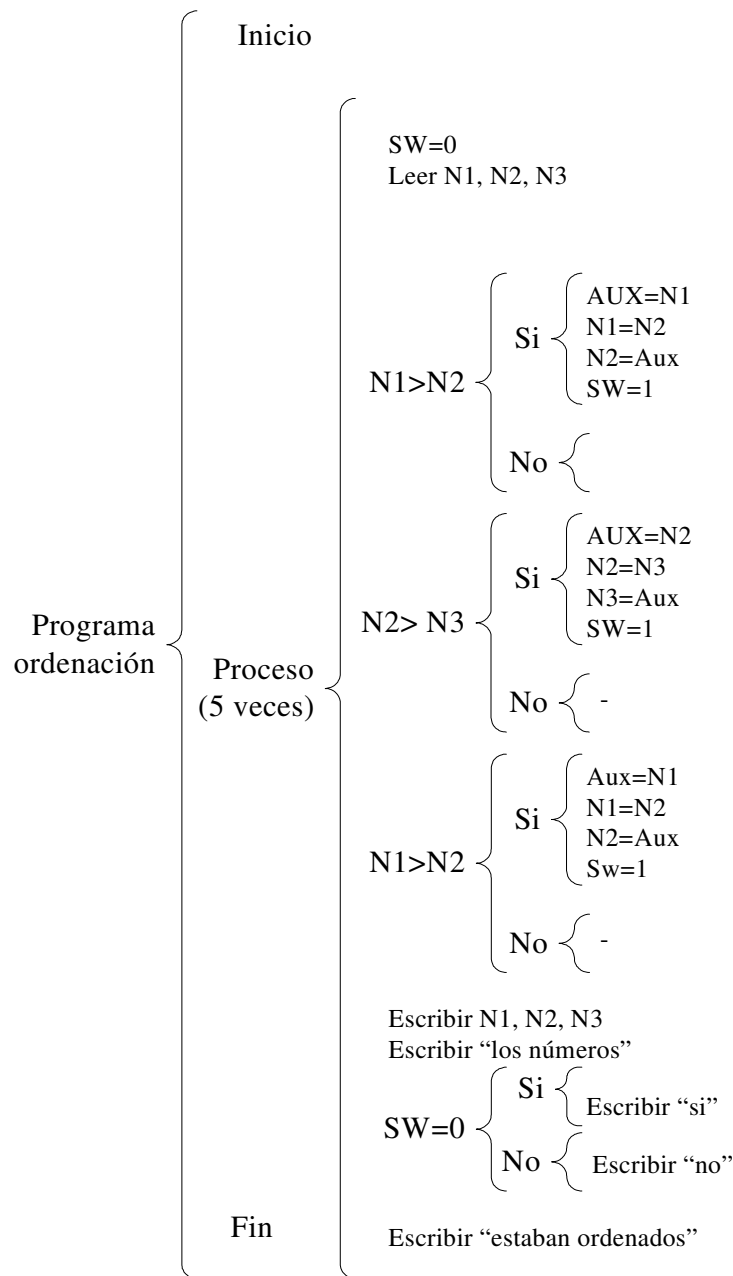


Figura 7. Diagrama de Warnier/Orr. Ejemplo de programa para ordenar números

El mismo Alcalde en [3], señala que un programa bajo el método de Jackson se desarrolla en cinco fases o pasos que se realizan consecutivamente y son las siguientes:

1. Definir las estructuras de datos. Consiste en una representación jerárquica de los datos definidos en las especificaciones del problema.
2. Encontrar correspondencia entre las estructuras de datos. Determinar qué entidades se pueden procesar simultáneamente, es decir que elementos ocurren el mismo número de veces y en la misma secuencia.

3. Formar la estructura del programa. Trata de reunir todas las estructuras de datos basándose en las correspondencias obtenidas para configurar la estructura del programa.

4. Listar y asignar las operaciones y condiciones a realizar. Consiste en listar las operaciones a realizar asignándolas a la estructura del programa y determinar las condiciones de alternativas y repeticiones.

5. Escribir la lógica esquematizada. Descripción del programa en el pseudo código propio de la metodología a partir del árbol obtenido en la fase anterior.

El diagrama de Jackson se trata de una simbología en forma de árbol y está formada por:

- ✓ Definición detallada de los datos de entrada y salida incluyendo los archivos lógicos utilizados.
- ✓ Representación del proceso o algoritmo.

La lectura del diagrama se hace en preorden (raíz, izquierda, derecha) y las reglas de construcción son las siguientes:

Regla 1. Se crea una componente del programa por cada correspondencia encontrada entre las entidades de las estructuras de datos, manteniendo las relaciones jerárquicas que existan entre ellas.

Regla 2. Añadir una componente de proceso a la estructura del programa por cada entidad de datos que no posea correspondencia en las estructuras de datos de entrada, manteniendo la relación entre sus elementos.

Regla 3. Añadir un componente de proceso a la estructura del programa por cada entidad de datos que no posea correspondencia en las estructuras de datos de salida, manteniendo sus relaciones y asegurando que los datos necesarios para producir la salida se encuentren presentes.

Los símbolos básicos utilizados por Jackson se muestran en la figura 8 y las estructuras de control que propone se encuentran en la figura 9. El cuadro simple representa un proceso común. El señalado con una O representa una operación opcional dependiendo de una condición. El cuadro con un * representa una repetición.



Figura 8. Símbolos de Jackson

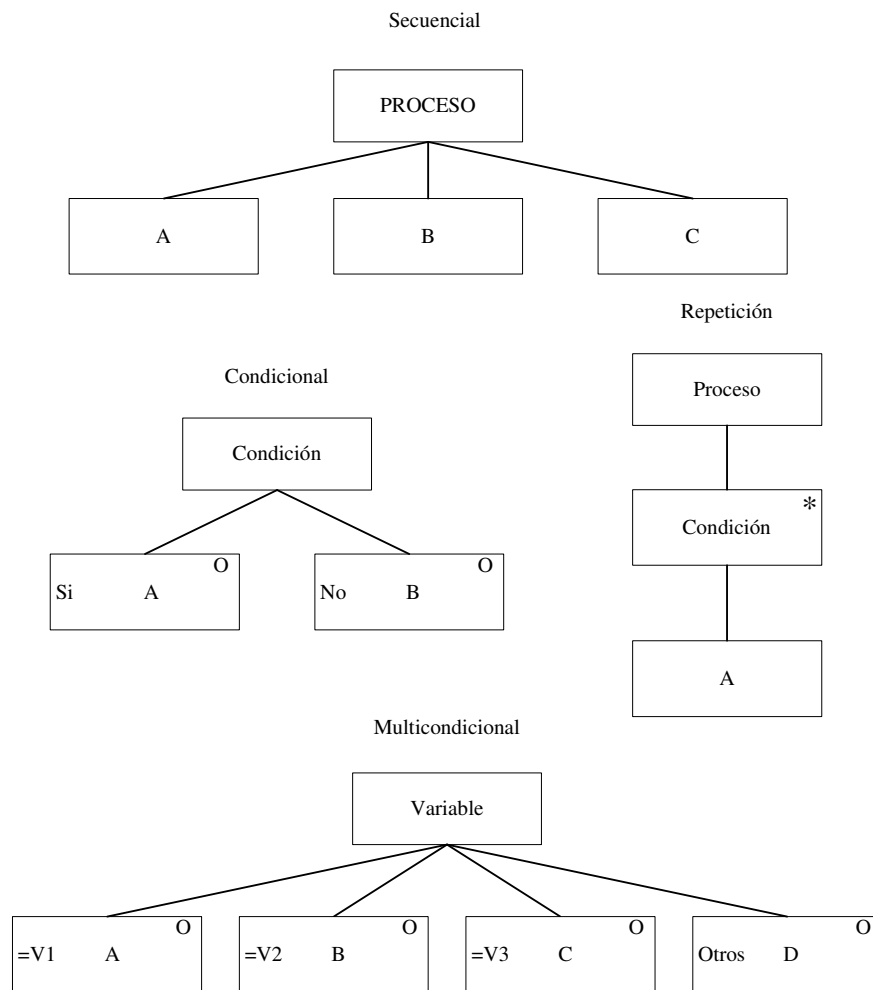


Figura 9. Estructuras de control de Jackson

La figura 10 presenta, con la simbología de Jackson, el mismo algoritmo de leer tres números, imprimirlos ordenados ascendentemente e indicar si originalmente estaban ordenados. Esto debe repetirse cinco veces.

La metodología de Bertini consiste en la descomposición de un problema en niveles, teniendo cada uno de ellos un inicio, un conjunto de procesos y un fin. Esta metodología representa la estructura de los programas y no las operaciones del tratamiento. Según Bertini, el diagrama también es un árbol pero se recorre en preorden inverso (raíz, derecha izquierda) y las instrucciones se ejecutan de derecha a izquierda pero el programador puede leerlo al revés si le resulta más cómodo.

Los símbolos básicos de Bertini son el rectángulo y el círculo. El inicio y el fin se señalan con un rectángulo cuyos extremos son medios círculos. Los diagramas constan de:

- ✓ Definición detallada de los datos de entrada y salida, incluyendo los archivos utilizados.
- ✓ Representación del proceso o algoritmo.

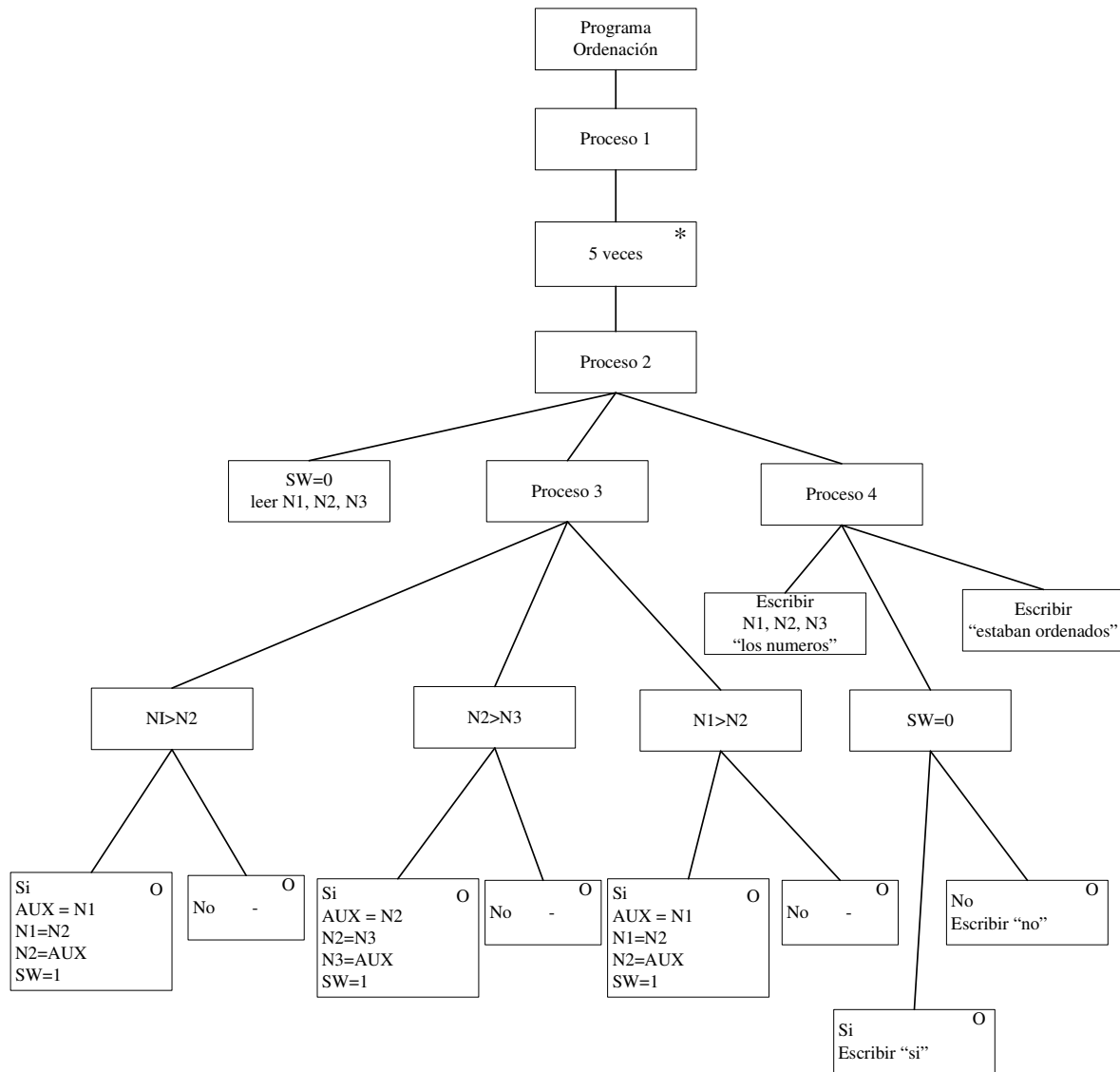


Figura 10. Diagrama de Jackson

La figura 11 presenta la sintaxis propia de Bertini para implementar las estructuras de control básicas de Jacopini. Para la modalidad secuencial Bertini permite tanto el uso de un único rectángulo como de varios nodos del árbol ejecutados de derecha a izquierda.

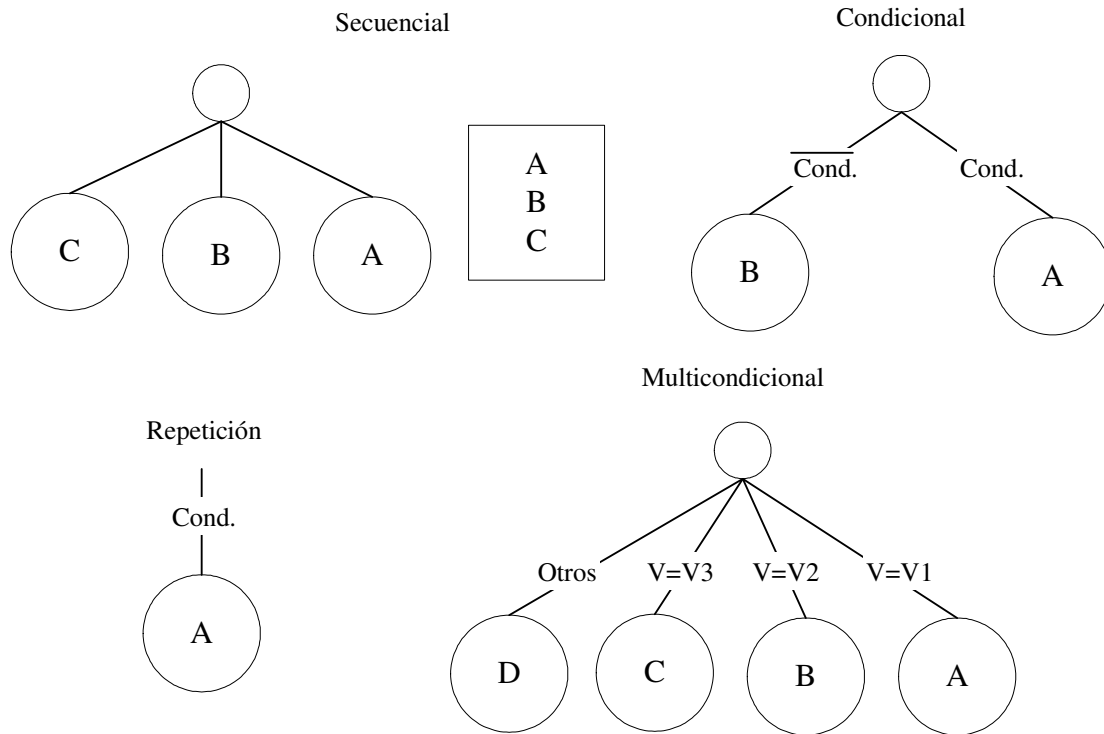


Figura 11. Estructuras de control de Bertini

En la figura 12 se muestra el mismo algoritmo ejemplo de leer tres números, los imprima ordenados ascendentemente e indique si originalmente estaban ordenados repitiéndolo cinco veces utilizando el modelo de Bertini.

El método de Tabourier también utiliza una representación en forma de árbol pero incluye más símbolos. Consta de:

- Definición detallada de los datos de entrada y salida, incluyendo los archivos utilizados.
- Representación del proceso o algoritmo.

La simbología utilizada se basa en rombos y rectángulos y la lectura se hace recorriendo el árbol en preorden (raíz, izquierda, derecha). Cada subárbol se recorre igualmente hasta llegar a las hojas o nodos terminales.

La figura 13 muestra las estructuras de control básicas del modelo de Tabourier.

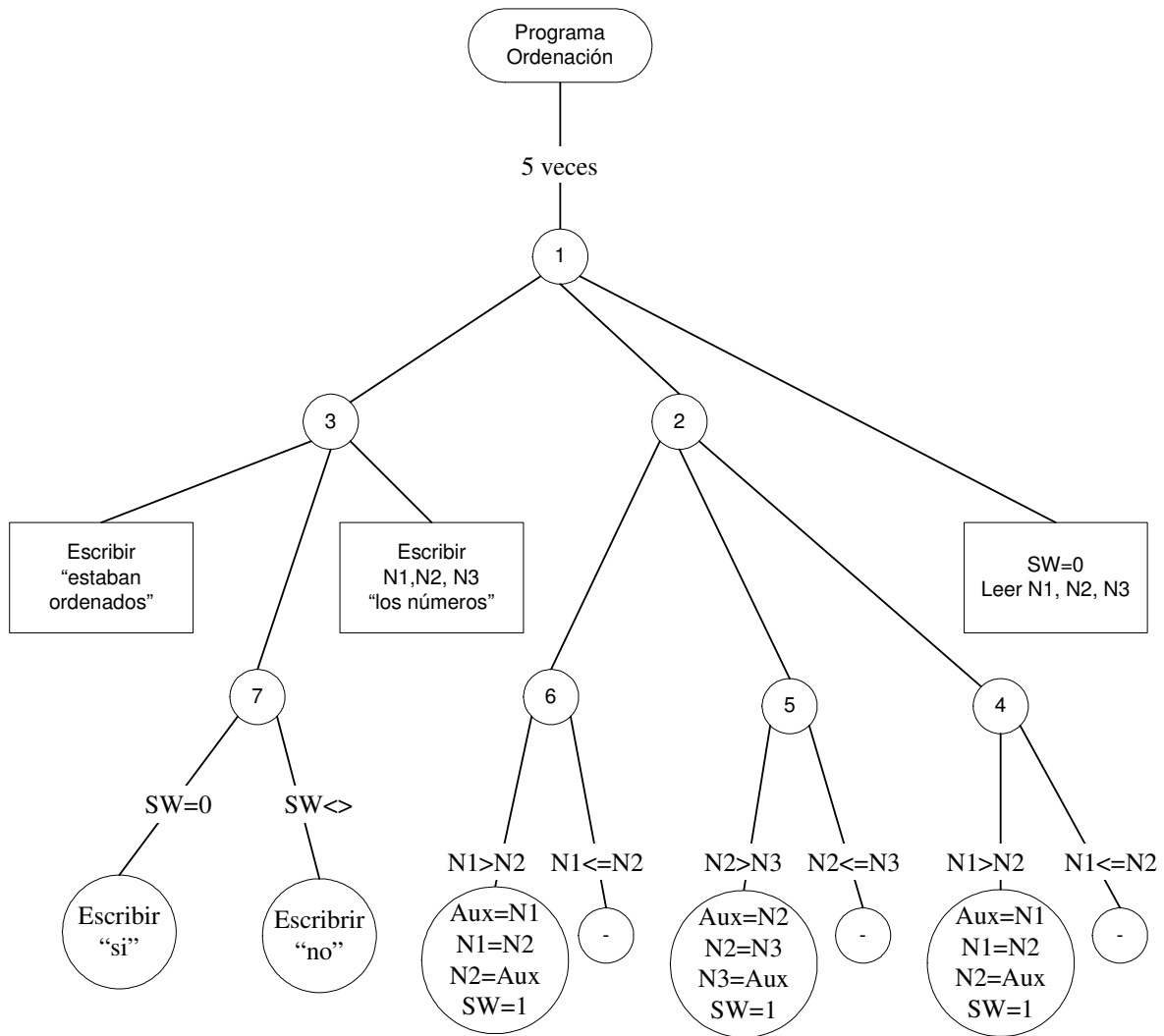


Figura 12. Diagrama de Bertini

La figura 14 muestra el desarrollo del algoritmo de ejemplo utilizando un diagrama de Tabourier.

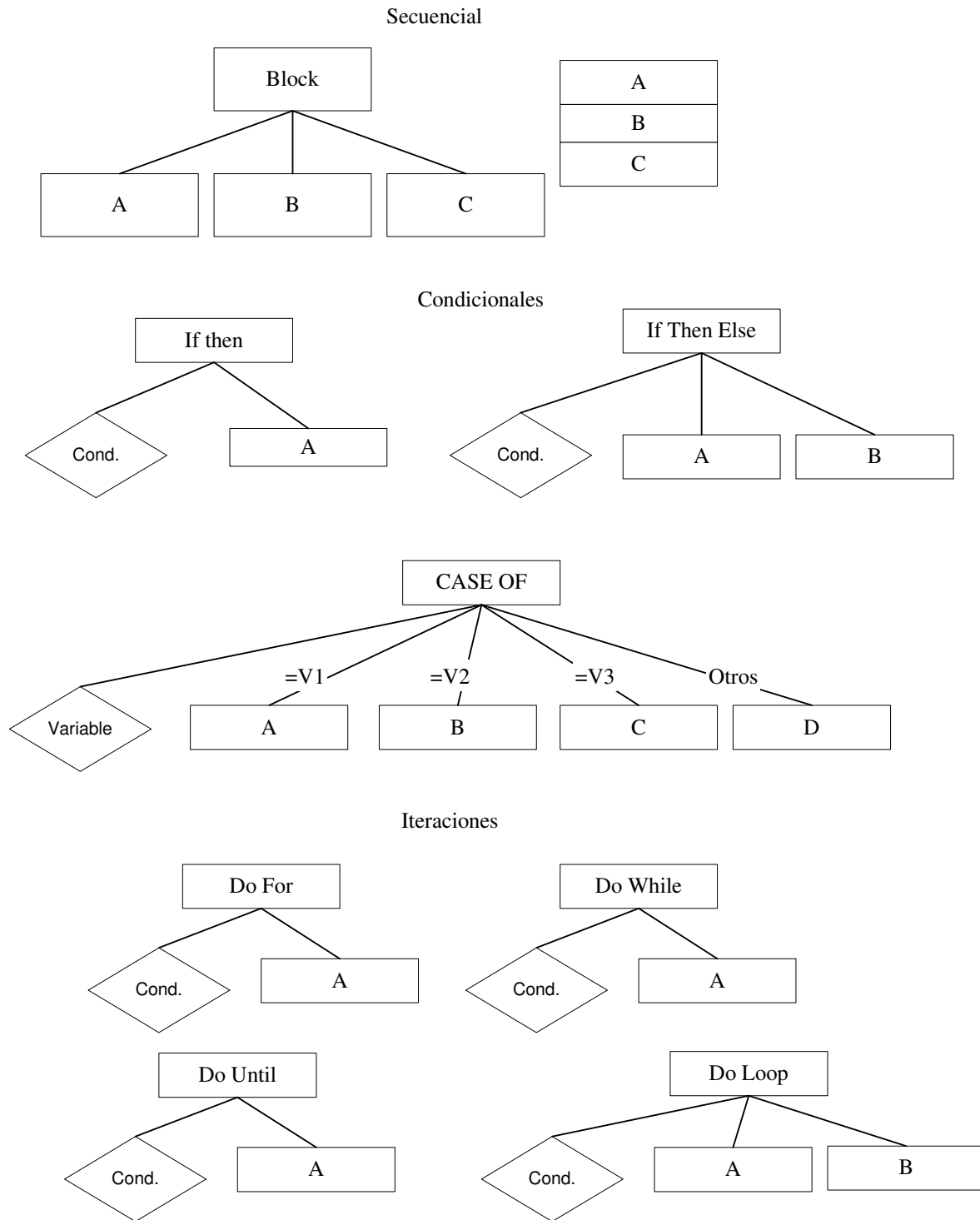


Figura 13. Estructuras de control de Tabourier

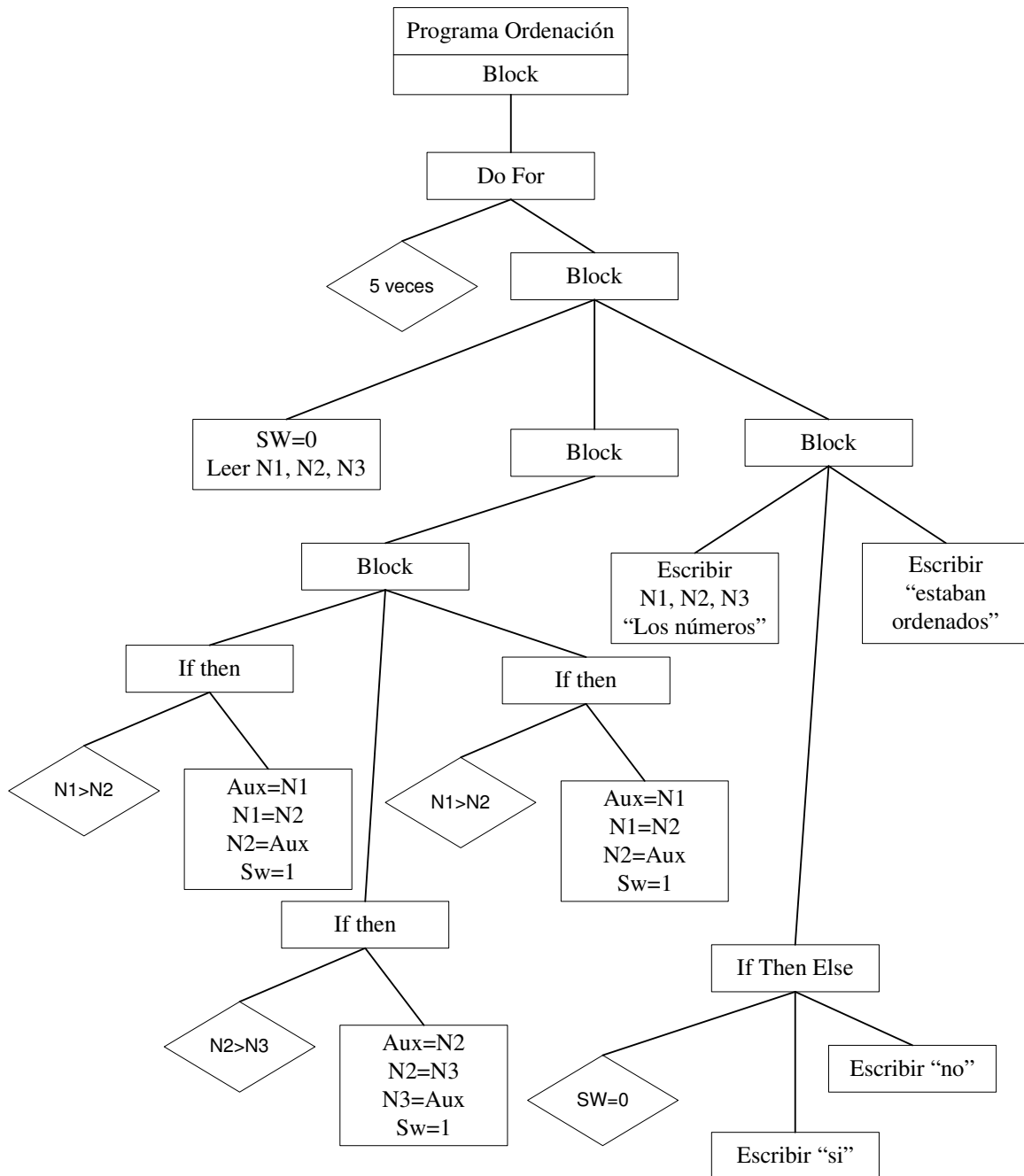


Figura 14. Diagrama de Tabourier

Los Diagramas de HIPO son gráficos más que descripciones en forma de prosa de un sistema. Ayudan al analista a contestar tres preguntas fundamentales:

1. ¿Qué hace el sistema o módulo?
2. ¿Cómo lo hace?
3. ¿Cuáles son las entradas y salidas?

Una descripción HIPO para un sistema consiste en una tabla visual de contenidos y de los diagramas funcionales.

Las consideraciones para la correcta elaboración de descripciones HIPO son las siguientes:

Sobre la Entrada:

- 1 ¿cuál es la forma de los datos?, ¿en que forma se presentan? ¿cómo son?;
- 2 ¿cuántos datos hay?;
- 3 ¿qué variables se necesitan?;
- 4 ¿qué formulas se necesitan?;
- 5 ¿deben aceptarse solamente los datos dados?;

Sobre el Procesamiento:

- 6 ¿qué orden siguen las operaciones en los datos?;
- 7 ¿En que orden deben obtenerse esos resultados?;
- 8 ¿debe efectuarse el procesamiento más de una vez?;

Sobre la Salida:

- 9 ¿qué resultados se necesitan?
- 10 ¿cómo deben ser los resultados?

La sintaxis de HIPO como lo explica Shelly en [93] se escribe en forma de prosa dentro de un formato. No hay reglas específicas de cómo debe escribirse una estructura de control determinada. En las tablas 1 a 4 se muestra un formato HIPO de ejemplo para resolver el algoritmo de leer tres números, los imprima ordenados ascendentemente e indique si originalmente estaban ordenados. Esto debe repetirse cinco veces.

Programa: Ordenación de Números		Fecha: sep 2004
Módulo: Principal		Referencia: 0001
Entrada	Proceso	Salida
	Repetir cinco veces el proceso de ordenación. Ref 0002	

Tabla 1. HIPO módulo principal

Ésta última metodología es originaria del paradigma libre y resulta muy notorio el hecho de que no formaliza las estructuras de control tal como lo propone el modelo estructurado, sin embargo, en el ejemplo se trata de representar el mismo algoritmo de ordenar tres números cinco veces.

De todos estos métodos para el diseño de algoritmos ninguno se conserva en la actualidad. Sin embargo, es importante rescatar que en el momento de diseñar la implementación de un método la aproximación por refinamiento, la identificación de los datos de entrada, salida (indicados como parte de la interfaz) y el manejo de las estructuras de datos internas siguen siendo ideas valiosas que bien podrían ser modernizadas.

Programa: Ordenación de Números		Fecha: sep 2004
Módulo: Proceso de Ordenación		Referencia: 0002
Entrada	Proceso	Salida
	SW=0	
N1, N2, N3	Si N1> N2 ejecutar el proceso A. Ref. 0003	
	Si N2> N3 ejecutar el Proceso B. Ref. 0004	
	Si N1> N2 ejecutar el proceso C. Ref. 0005	
		N1, N2, N3
		“los números”
	Si SW = 0 Entonces	“si”
	De lo contrario	“no”
		“Estaban Ordenados”
Programa: Ordenación de Números		Fecha: sep 2004
Módulo: Proceso A		Referencia: 0003
Entrada	Proceso	Salida
	AUX=N1	
	N1=N2	
	N2=Aux	
	SW=1	

Tabla 2. HIPO Proceso de ordenación

Programa: Ordenación de Números		Fecha: sep 2004
Módulo: Proceso B		Referencia: 0004
Entrada	Proceso	Salida
	AUX=N2	
	N2=N3	
	N3=Aux	
	SW=1	

Tabla 3. HIPO Proceso alternativo B

Programa: Ordenación de Números		Fecha: sep 2004
Módulo: Proceso C		Referencia: 0005
Entrada	Proceso	Salida
	AUX=N1	
	N1=N2	
	N2=Aux	
	SW=1	

Tabla 4. HIPO Proceso alternativo C

Dentro del seno del paradigma orientado a objetos sobresale la propuesta de Droomey [36]. Sin embargo, es notorio el poco desarrollo que se tiene en este particular un tanto motivado por la intención de distintos autores de apartarse de todo lo que fuese estructurado alegando que fomentaba malos hábitos en programación.

Como resultado del auge del paradigma orientado a objetos una gran cantidad de técnicas de análisis y diseño surgen concluyendo con la aceptación general de UML como lenguaje de modelado. Este lenguaje de modelado propone simbología para diagramas de secuencia, diagramas de actividad y diagramas de estado, sin embargo, no existe en él una simbología para diseñar los procesos internos de los métodos (lo más cercano son los contratos) ni tampoco una simbología para representar el teorema de la estructura a pesar de la obvia necesidad de ello.

2.2.4. Herramientas y Técnicas

Todas las metodologías de diseño de algoritmos han utilizado herramientas diversas. De hecho algunas son, en sí mismas, técnicas de diseño de algoritmos.

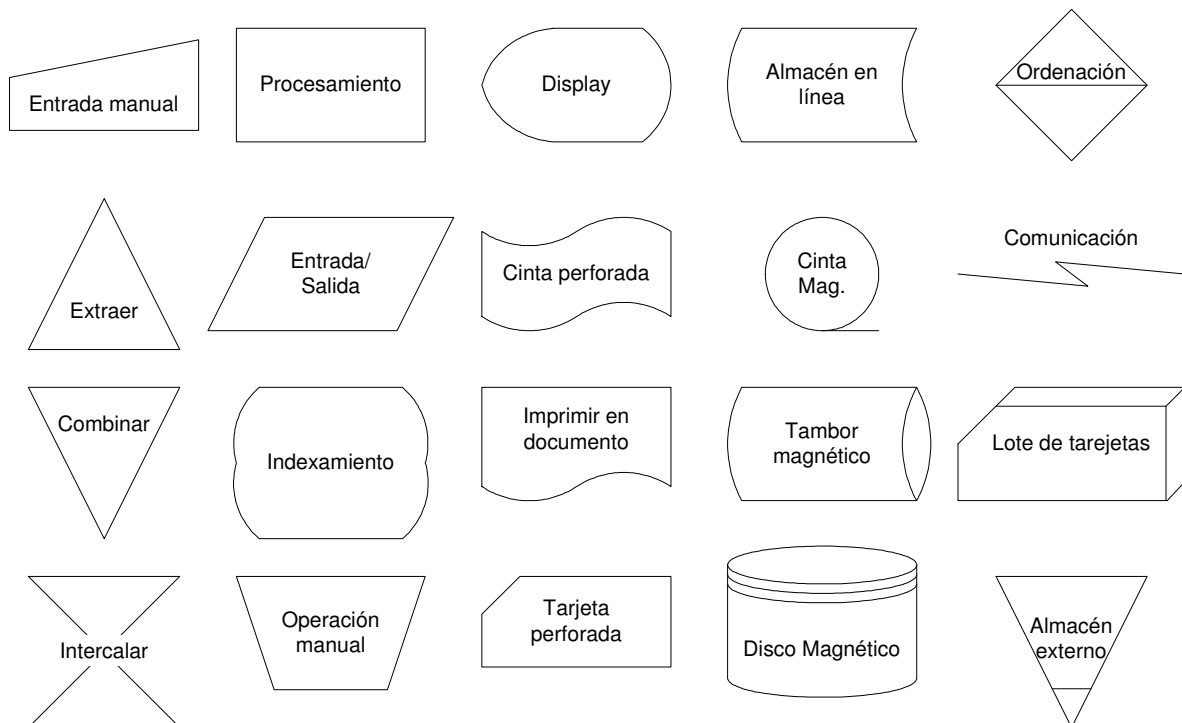


Figura 15. Símbolos utilizados para los diagramas de flujo propuestos por IBM

Dentro del paradigma libre aparecieron con uso generalizado los Diagramas de Bloques, Diagramas de Flujo, Grafos, Tablas de Verdad, Tablas de Decisiones, Árboles de decisiones y pseudocódigo. Cada uno con sus reglas y características propias.

Los diagramas de bloques se utilizan para describir conceptualmente las grandes acciones que se deben realizar por un algún algoritmo en particular. No detalla los pasos, más bien los describe en términos globales por lo que su uso en el diseño algorítmico fue limitado.

Los diagramas de flujo detallan a los diagramas de bloques. Cada paso del algoritmo está asociado directamente con un símbolo y se encuentran implementadas las estructuras de secuencia, repetición y saltos condicionales e incondicionales. IBM [93] utilizaba los símbolos de la figura 15.

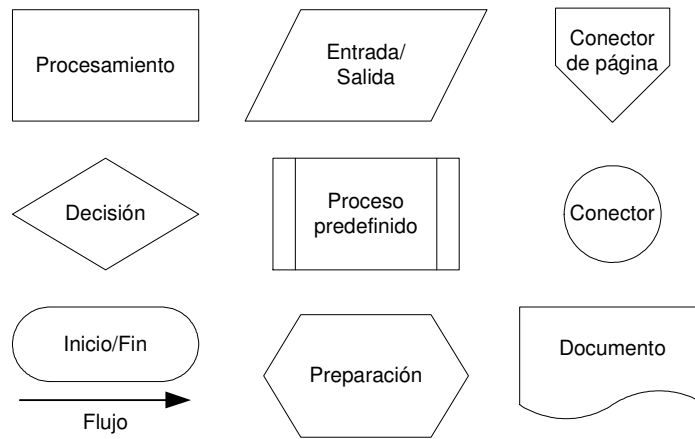


Figura 16. Símbolos de diagramas de flujo ANSI

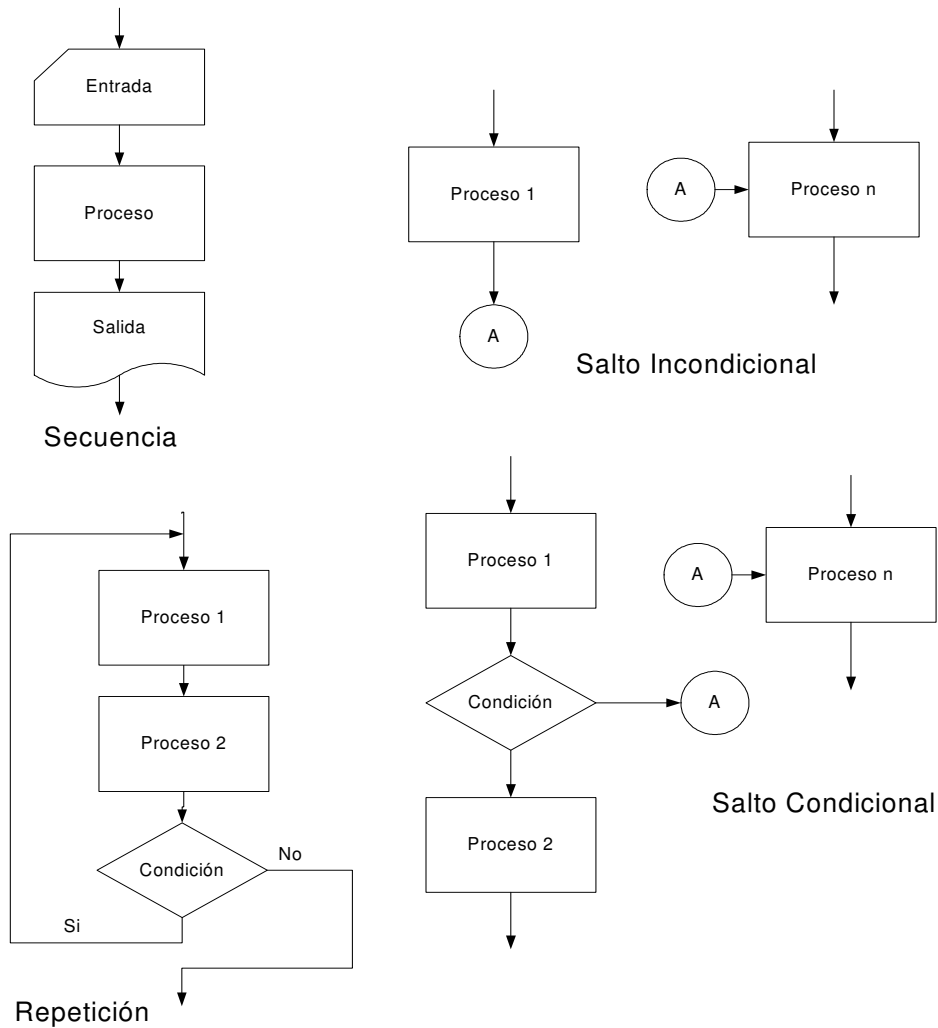


Figura 17. Estructuras de control del paradigma libre

La figura 16 muestra los símbolos estandarizados por la ANSI para los diagramas de flujo [93]. Y la figura 17 muestra las estructuras de control [93] utilizadas durante el paradigma libre.

En la figura 18 se muestra un ejemplo de un diagrama de flujo utilizando las estructuras de control que predominaban en el paradigma libre. El ejemplo fue obtenido de [39] y presenta un algoritmo para obtener los factores primos de un número entero dado por el usuario. Obsérvese en el ejemplo el uso de saltos incondicionales y condicionales.

Los grafos se utilizan para crear máquinas de estados y las acciones que motivan los cambios de estos. Tienen otros usos variados también para crear la secuencia de acción de un algoritmo. En la figura 19 se muestra un ejemplo expuesto en [70]. El modelo expone un modelo matemático que señala que si hay un incremento en P (Población) se incrementará G (Basura) lo que incrementará a su vez B (Bacterias), lo que a su vez incrementará D (Enfermedades) y eso finalmente disminuirá P (Población). Sin embargo, al agregarse otros factores como M (modernización) y éste se incrementa, atraerá más C (Migración a Ciudades) lo que también incrementará P; pero simultáneamente un incremento en M significaría un incremento en S (Saneamiento) lo que disminuiría B (Bacterias) y disminuiría D (Enfermedades).

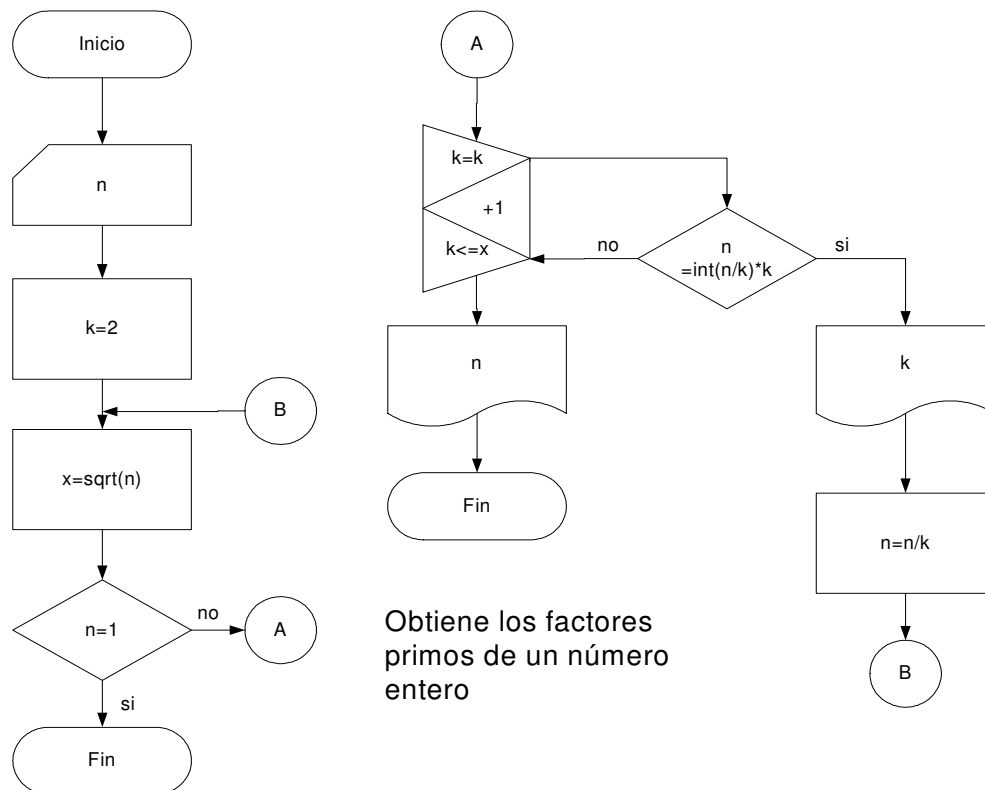


Figura 18. Ejemplo de diagrama de flujo en el paradigma libre.

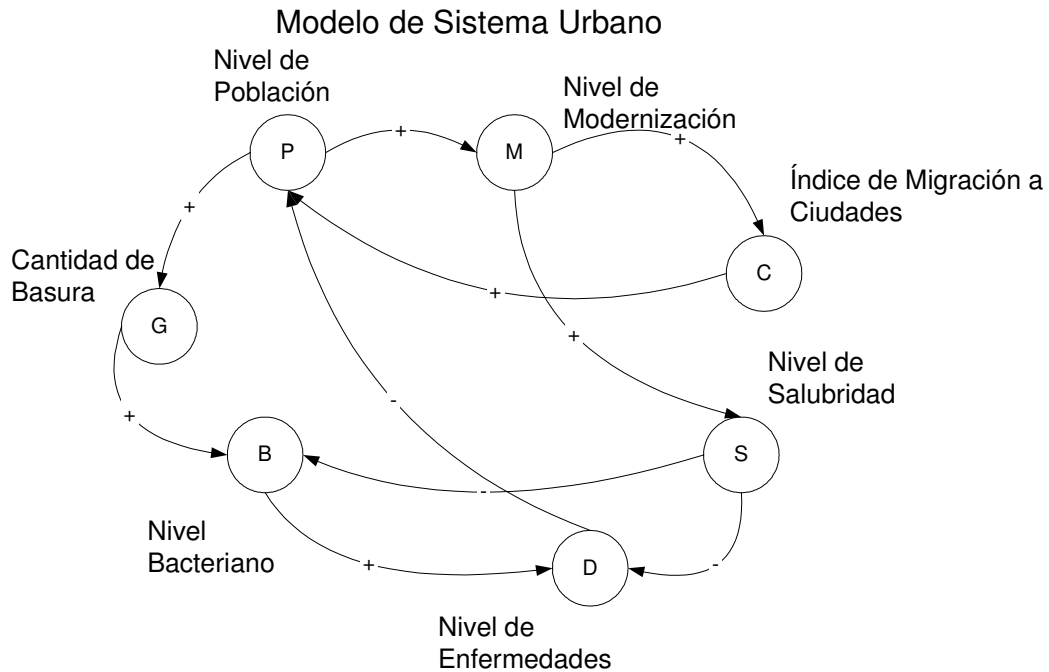


Figura 19. Ejemplo de grafo

Las tablas de verdad se utilizan para evaluar funciones booleanas [75] las cuales se construyen utilizando operadores relacionales (<, >, <=, >=, =, <>) y operadores lógicos (and, or, not, xor). El dominio de evaluar una función booleana es verdadero o falso, gracias a ello son la base de los ciclos controlados y los saltos condicionales; más adelante, en el paradigma estructurado se utiliza también para los ciclos y para las estructuras condicionales.

Las tablas de decisión, las explica Alcalde en [3], son una representación tabular de la lógica de un problema en el que se presentan variadas situaciones y diferentes alternativas para cada una de ellas. Está dividida en cuatro cuadrantes. Los dos de la izquierda (superior e inferior) enlistan las condiciones y las acciones respectivamente. Los dos de la derecha señalan reglas de decisión indicando que valores tomas las condiciones (entradas - en la parte superior) y que acciones se llevan a cabo (salidas - en la parte inferior). Ver la figura 20.

Condición 1	s	s	s	s	n
Condición 2	s	n	s	n	n
Condición 3	n	s	s	n	n
Acción 1	x			x	
Acción 2		x		x	x
Acción 3			x		x

Figura 20. Tabla de decisión

Los árboles de decisiones, como las explica Senn en [91], son estructuras arborescentes que se bifurcan en función de la condición que evalúan, cada rama resultante puede a su vez ser

evaluada en una nueva condición y así sucesivamente. Las hojas del árbol contienen las acciones a seguir. Ver figura 21.

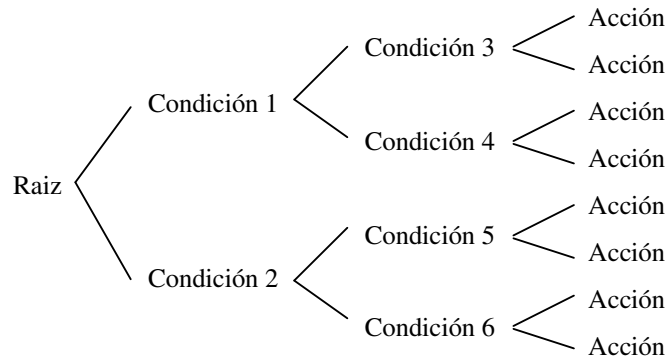


Figura 21. Árboles de decisiones

El pseudocódigo [3] [45] [55] es una notación mediante la cual se puede escribir la solución de un problema en forma de algoritmo dirigido a la computadora utilizando frases del lenguaje natural sujetas a ciertas reglas. Con él se puede expresar la lógica de un programa, es decir, su flujo de control y las operaciones que tiene que realizar. Entre sus características debe permitir la descripción de:

- ✓ Instrucciones de entrada/salida
- ✓ Instrucciones de proceso
- ✓ Sentencias de control del flujo de ejecución
- ✓ Acciones compuestas que se refinan posteriormente.

En el siguiente ejemplo se muestra el algoritmo de push a un stack utilizando pseudocódigo:

```
Si Top > MaxCapacidad
Entonces
    Imprime "error de overflow"
    Fin de ejecución
Top = Top + 1
Leer Stack(Top)
Fin de ejecución
```

Durante el auge del paradigma estructurado se continuó con la mayoría de las ya existentes y se agregaron, o consolidaron, otras como el mismo pseudocódigo (al que se le incluyeron las estructuras de control de Jacopini), los ordinogramas [3] (que son los diagramas de flujo pero ahora con reglas de estructura para su elaboración) y los diagramas de Chapin (Nasi/Shneiderman) [3] [55].

A continuación se presenta la solución al ejemplo de ordenar tres números que se ha venido presentando en pseudocódigo pero incluyendo las estructuras de control de Jacopini:

```
Inicio del Programa
Repite 5 veces
```



```

SW=0
Leer N1, N2, N3
Si N1>N2 entonces
    Aux=N1
    N1=N2
    N2=Aux
    SW=1
Si N2>N3 entonces
    Aux=N2
    N2=N3
    N3=Aux
    Sw=1
Si N1>N2 entonces
    Aux=N1
    N1=N2
    N2=Aux
    SW=1
Escribir N1, N2, N3
Escribir "Los números "
Si SW=0 entonces
    Escribir "si"
De lo contrario
    Escribir "no"
Escribir "Estaban Ordenados"
Fin Repite
Fin del Programa
    
```

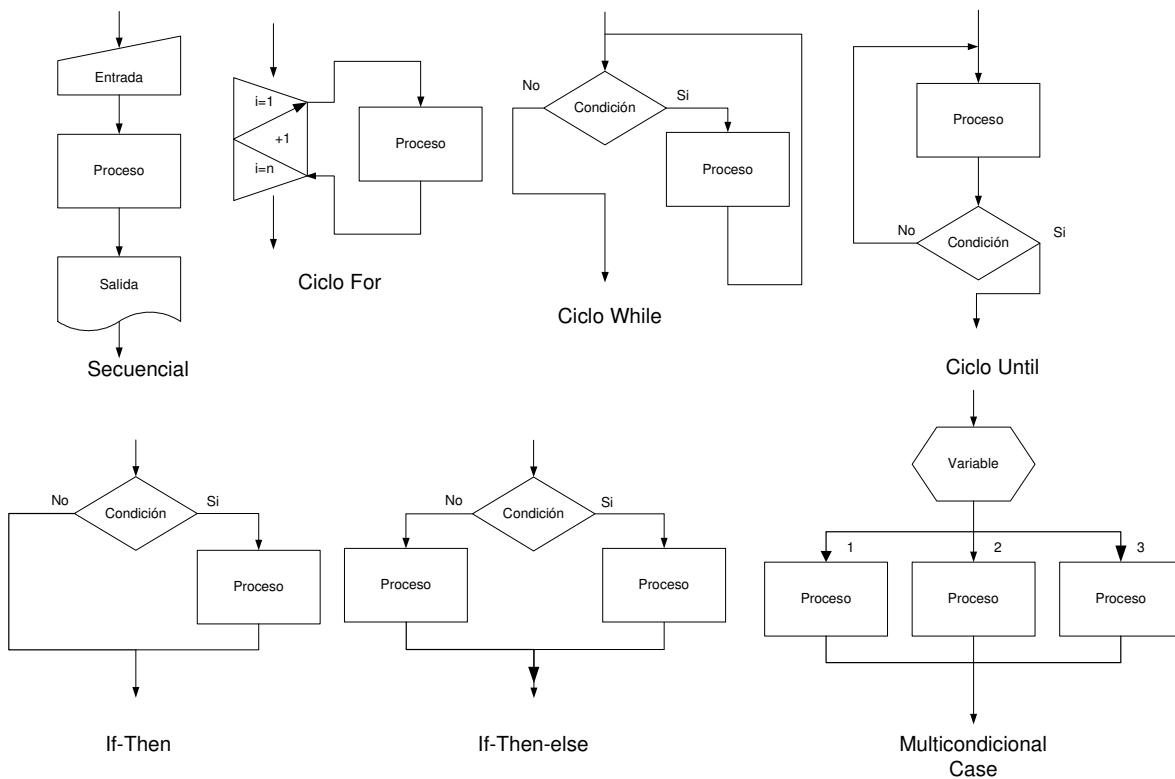


Figura 22. Estructuras de control en ordinogramas

Los ordinogramas son los diagramas de flujo pero ahora utilizando las reglas del teorema de Jacopini para convertirse en diagramas estructurados. Los ordinogramas aspiran a evitar los saltos de control tanto condicional como incondicional e incluso evitan romper las estructuras mismas. Sin embargo, la propia simbología favorece estas violaciones que pretende evitar debido a que líneas de secuencia de ejecución (flechas) podrían fácilmente violar la estructura señalando a algún lugar que no es el que la estructura exige respetar. En la figura 22 se muestran las estructuras tal y como deben ser operadas. La figura 23 muestra el algoritmo de ordenación de tres números utilizando ordinogramas. Obsérvese como las estructuras de control son escrupulosamente vigiladas en su uso.

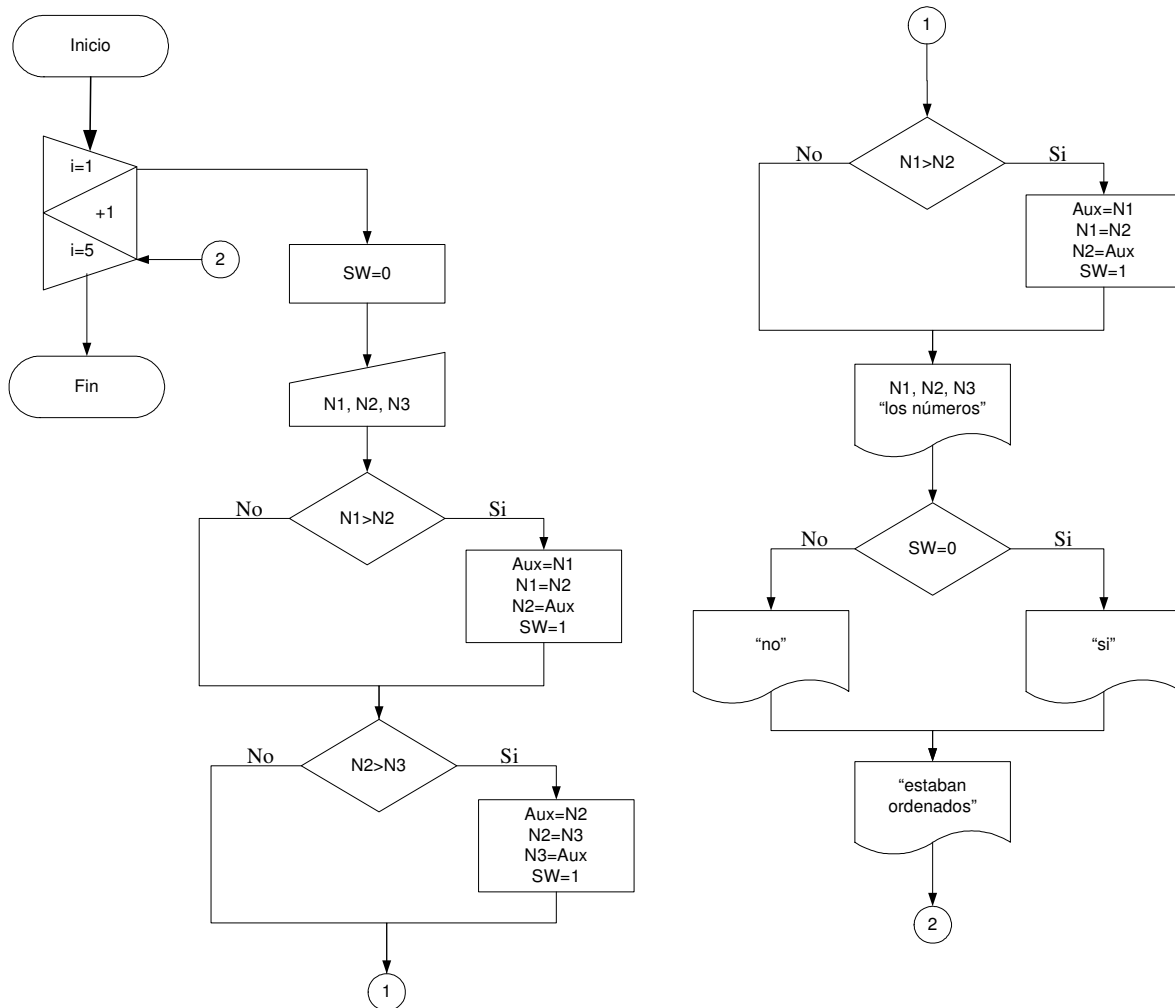


Figura 23. Ordinograma

Los diagramas de Chapin o también conocidos como Nassi/Shneiderman están formados por una serie de rectángulos de diversas formas que representan las estructuras. Estos rectángulos se anidan y enlazan para su conformación. Las acciones sucesivas se escriben en cajas sucesivas. No incluyen flechas. Permiten una visión estructurada que aspira facilitar su traducción a algún lenguaje.

En la figura 24 se puede observar la sintaxis en los diagramas Nassi/Shneiderman Para representar las diversas estructuras de control. En la figura 25 se muestra el algoritmo de ordenación de tres números utilizando este diagrama.

Durante el paradigma orientado a objetos no hay surgimiento de herramientas nuevas sobre el diseño de algoritmos más que el diagrama de actividades [40] que originalmente no era parte de UML aunque se incluyó posteriormente. En la figura 26 se puede observar la sintaxis de este diagrama misma que incluye la secuencia, la condición pero no incluye sintaxis para iteración en el sentido imperativo por lo que no se considera completamente adecuado para el diseño algorítmico. Por otra parte, incluye sintaxis para el fork y el join que no existe en las técnicas anteriores pero no incluye la recursividad en su sintaxis.

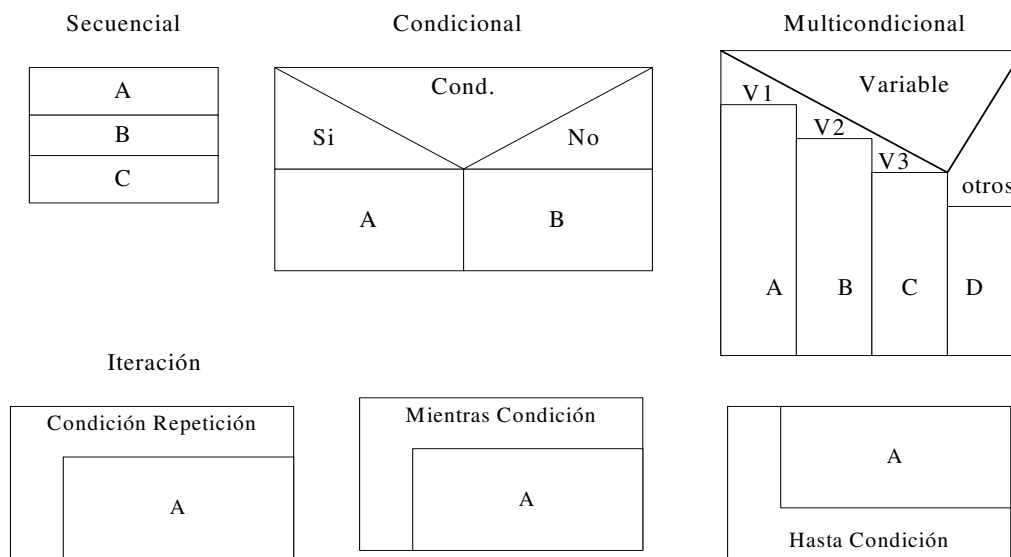


Figura 24. Estructuras de control de Chapin (Nassi/Shneiderman)

Es interesante observar que muchas de las herramientas para el diseño de algoritmos aparecen en el modelo de programación libre, quizás por la ausencia de estructuras de control formales ante una necesidad real de diseño. En la fase de la programación estructurada nacen algunas nuevas y la mayoría de las anteriores prevalecen evolucionando, adecuándose e incluyendo las nuevas reglas que las teorías vigentes exigían. Con el paso de los años, únicamente el pseudocódigo sobrevivió hasta la programación orientada a objetos sin que surgieran nuevas herramientas o métodos lo cual alerta de la necesidad de fortalecer o actualizar las técnicas en este particular.

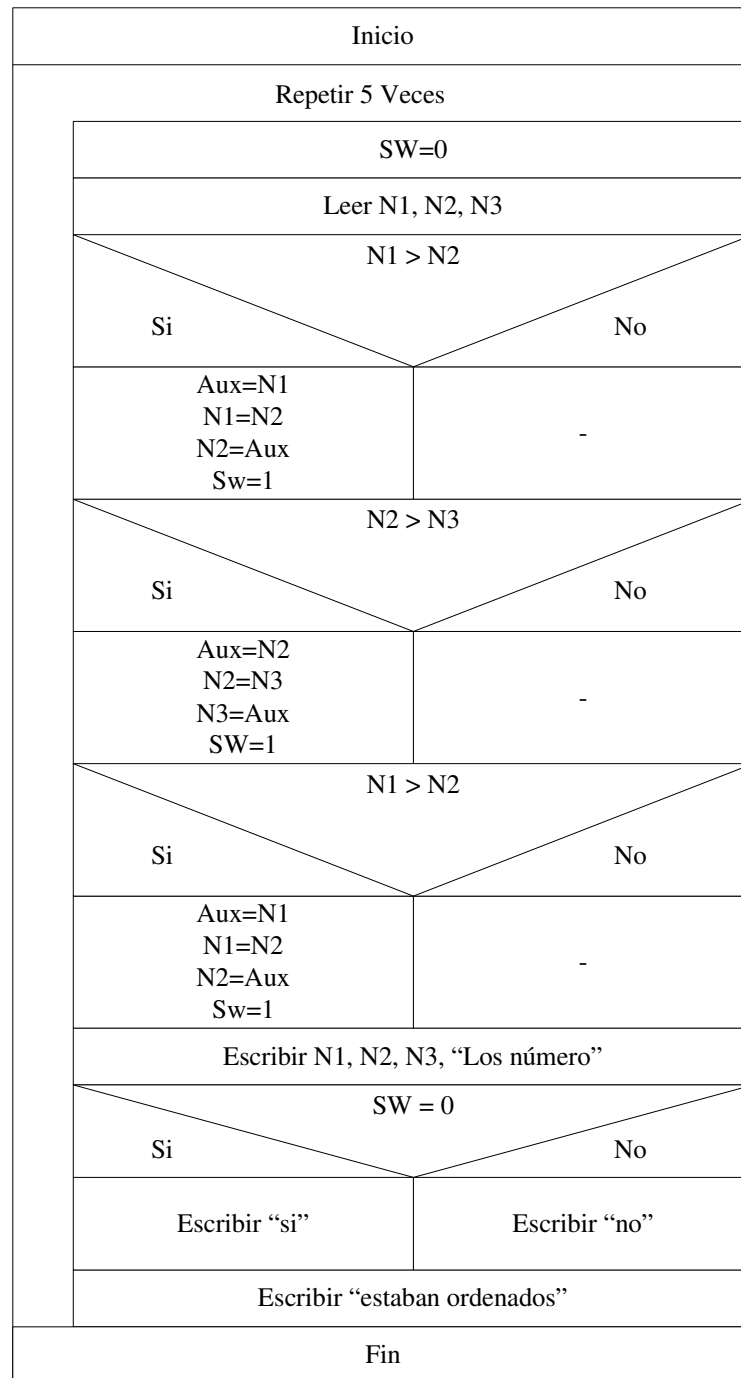


Figura 25. Diagrama de Chapin (Nassi/Shneiderman)

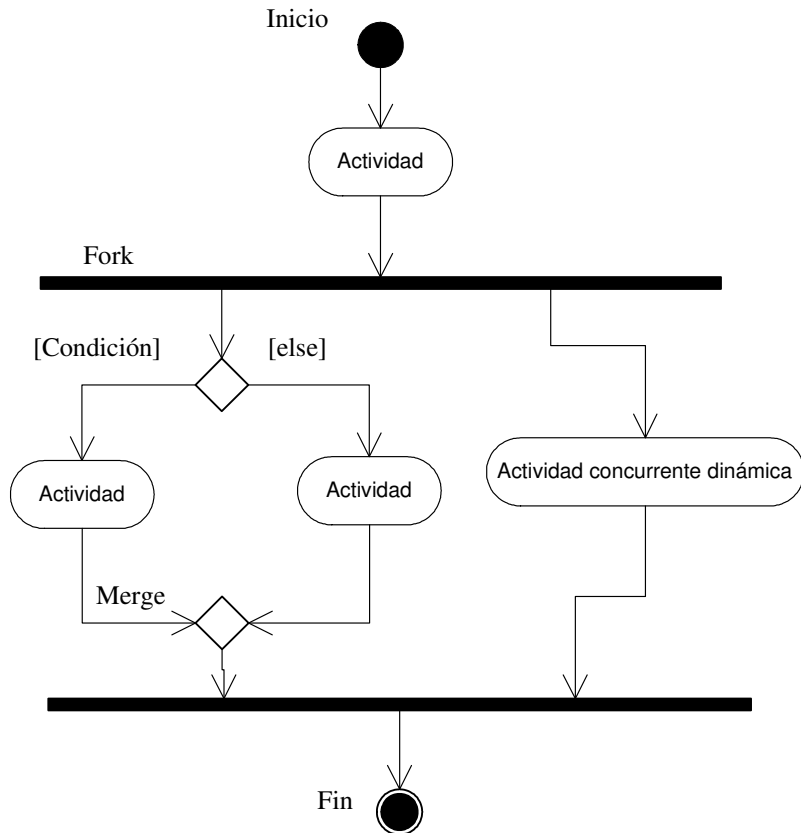


Figura 26. Diagrama de actividad de UML

2.2.5. Heurísticas

Las heurísticas son directrices, consejos o recomendaciones que se sugieren para obtener mejores resultados en una acción de invención. En el caso del diseño de programas Pressman [86] expone siete heurísticas para crear la estructura de un programa.

1. Evaluar la "primera iteración" de la estructura del programa para reducir el acoplamiento y mejorar la cohesión.
2. Intentar minimizar las estructuras con mucho grado de salida; intentar concentrar a medida que aumenta la profundidad.
3. Mantener el alcance del efecto de un módulo dentro del alcance del control de ese módulo.
4. Evaluar las interfaces de los módulos para reducir la complejidad, la redundancia y mejorar la consistencia.
5. Definir módulos cuya función sea predecible, pero evitar módulos que sean demasiado restrictivos.
6. Intentar conseguir módulos de "entrada controlada", evitando "conexiones patológicas"
7. Empaquetar el software basándose en las restricciones del diseño de portabilidad.

Por su parte Ledin [63] presenta una serie de heurísticas mucho más enfocadas al diseño interno del algoritmo y son las siguientes:

1. Haga que el orden consecutivo sea la base fundamental de la estructura de control.
2. No sacrifique legibilidad por eficiencia.
3. Respete el proceso iterativo; considérela como una unidad.
4. Trazar un diagrama puede ayudar.
5. No haga saltos al interior de un proceso iterativo.
6. Inicialice las variables antes de usarlas.
7. Inicialice los contadores y sumadores a cero antes de usarlos.

Con el paso de los años, una gran cantidad de esfuerzo de la ingeniería de software se ha concentrado en las técnicas de modelado del dominio de la solución a un alto nivel de abstracción. Sin embargo, al parecer no se han hecho avances importantes en materia de modelado al interior de los métodos de las clases. El diseño por contrato [62], las pre y post condiciones [71] y los diagramas de estado [13] son muy útiles para delinear sus especificaciones pero quedan pobres en el momento de hacer el diseño imperativo.

2.2.6. Mapa Mental del Modelado Algorítmico

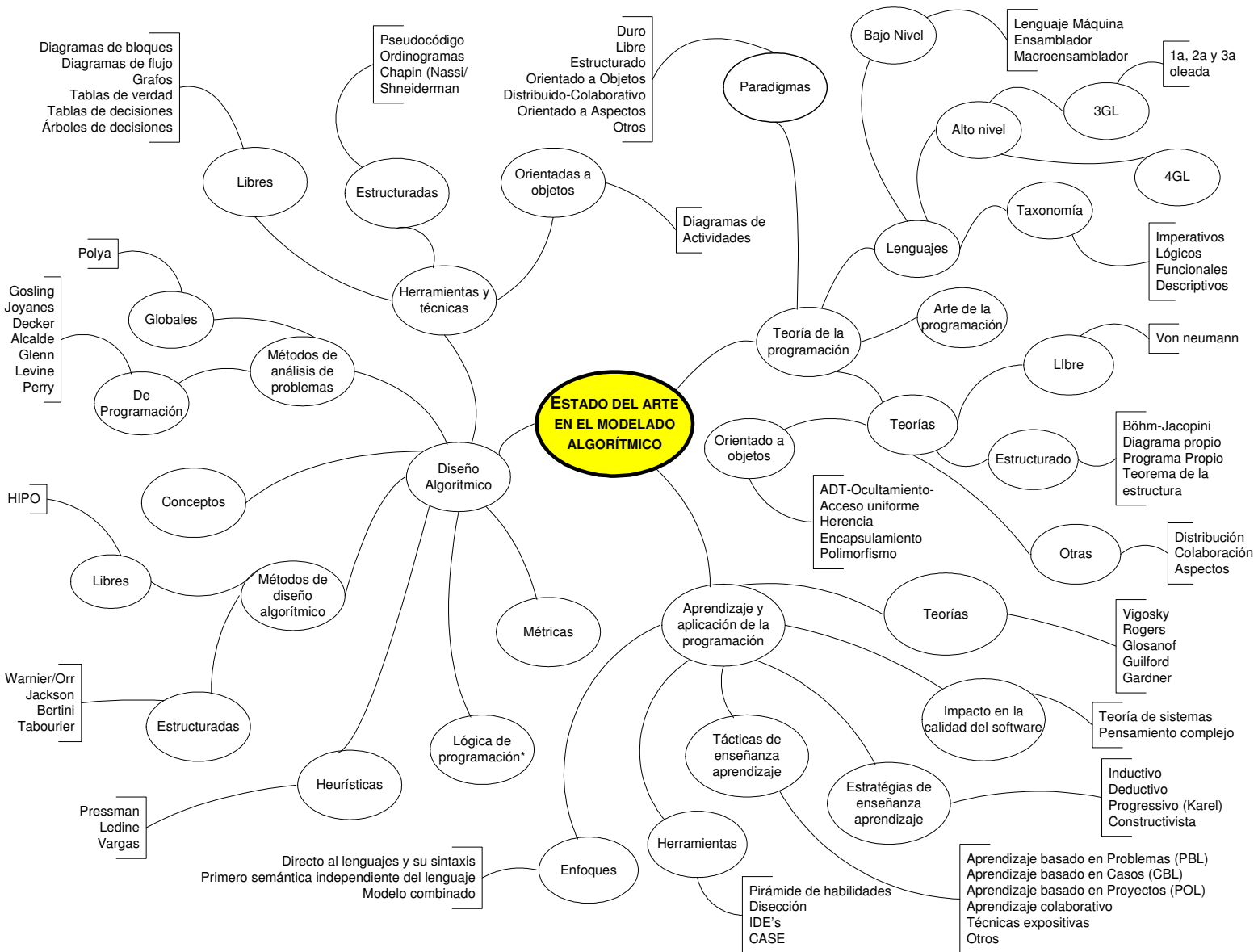


Figura 27. Modelado algorítmico

2.3 Aprendizaje y Aplicación de la Programación

2.3.1. Conceptos

Glenn [45] ha señalado de modo muy claro que la forma como se descubren algoritmos tiene una estrecha relación con la forma como se enseña la programación. De forma textual él señala: “Las técnicas de descubrimiento de algoritmos y estudios en esta rama de la computación se apoyan en áreas como la psicología humana de la resolución de problemas y las teorías de la educación.” Con esto en mente se abordarán las principales actividades que se han realizado a través del tiempo para la educación de profesionales de las Tecnologías de la Información ya que de ellas depende, en cierta medida, la capacidad de crear algoritmos, programas y con ellos la capacidad de resolver problemas.

La manera como los ingenieros forman a nuevos ingenieros va de la mano con las corrientes educativas del momento así como de la propia experiencia y juicio de lo que se considera bueno o malo, apropiado o inapropiado. Es interesante señalar que la “didáctica de la programación” - y sus sinónimos como metodología de la programación y otros similares - ha sido parcialmente atendida e incluso delegada a otras áreas del conocimiento y no a las ciencias de la computación.

2.3.2. Impacto en la Calidad del Software

Invocando una alegoría que se le atribuye a Mike Vance, directivo de la Walt Disney, que reza así: “Lo que somos es producto de lo hacemos; lo que hacemos es producto de lo que sabemos; y lo que sabemos es producto de lo que aprendemos.” Este principio subraya lo determinante que es el dominio de las cosas como factor de éxito. En particular el dominio de la programación como basamento de todo producto de software. Este es un claro ejemplo de cómo la teoría de sistemas señala las relaciones entre los distintos elementos de un fenómeno que es de naturaleza compleja (por la teoría de la complejidad más que por el adjetivo mismo).

Siguiendo la idea de Vance, se puede deducir que un producto de software (implicando su calidad) es resultado del seguimiento de una metodología y de los procesos de manufactura del mismo, es decir es producto de lo que se hace; las metodologías y procesos de manufactura del software es producto de cómo se planea y se ejecuta, es decir de lo que se sabe hacer. Esta capacidad de ejecución de tareas es resultado de lo que se aprendió en su oportunidad en las aulas.

Con animo holista es claro que la calidad del software está en dependencia de diversos factores pero que cada uno de ellos, a su vez, depende de lo que se sabe hacer y esto a su vez nos lleva a una clara conclusión: El grado de aprendizaje (dominio) que tiene un ingeniero de software sobre las herramientas, técnicas, métodos, etc. determina la calidad del producto de software.

2.3.3. Corrientes Educativas

A continuación se presenta el reporte del estado del arte de las principales corrientes educativas y sus exponentes.

2.3.3.1. Skinner

Ante la imposibilidad de saber con certeza lo que sucede dentro del cerebro, Burrhus F. Skinner decidió tomarlo como caja negra y preocuparse únicamente por las entradas (estímulos) y salidas (respuestas). Cuando la respuesta es correcta, se debe otorgar un refuerzo positivo.

Esta teoría llevó a Skinner a diseñar una máquina de enseñar. Este artefacto mecánico plantea una pregunta al alumno, quién debe presionar el botón que corresponda a la respuesta correcta. Si acierta, la lección continúa, pero si falla el ejercicio recomienza. “Cada información nueva suministrada por la máquina da lugar, así, a elecciones que testimonian la comprensión obtenida, con tantas repeticiones como sean necesarias y con un progreso ininterrumpido en caso de éxitos constantes. Cualquier disciplina puede, pues, programarse de acuerdo con ese principio, ya se trate de razonamiento puro o de simple memoria” [83].

2.3.3.2. Piaget

Jean Piaget es el creador del paradigma constructivista, cuya idea principal es que es estudiante construye activamente su propio conocimiento. La mente del estudiante transforma lo que recibe del mundo externo para determinar lo que se aprende. Es decir, el aprendizaje no es la recepción pasiva de la enseñanza, sino un trabajo activo de parte del estudiante, en el que el maestro juega un papel importante apoyando, cuestionando y actuando como modelo o entrenador [30].

Piaget encuentra cuatro métodos básicos de enseñanza [83]:

1. Método receptivo.- El profesor se encarga de dar la lección.
2. Método activo.- Adquisición de conocimientos por la acción. El alumno es activo “en el sentido de un redescubrimiento personal de la verdades por conquistar, haciendo recaer esta actividad en una reflexión interior y abstracta”.
3. Método intuitivo.- Se suministra a los alumnos representaciones audiovisuales de actividades, sin conducir a una realización efectiva de éstas.
4. Método programado.- Aprendizaje basado en estímulo-respuesta (ver Skinner 2.3.3.1). Puede enseñar un saber eficientemente, más no un razonamiento.

2.3.3.3. Vigotsky

En su teoría de aprendizaje social, Vigotsky define el aprendizaje como la internalización de conocimiento que ocurre cuando un proceso extrapersonal se transforma en un proceso intrapersonal individual [6]. El aprendizaje es un proceso complejo que se da en el contexto de la interacción entre medio ambiente y pensamiento. El estudiante es parte de un sistema, y su medio para interactuar consiste en los lenguajes [42].

Vigotsky plantea el concepto de la *zona de desarrollo proximal* de un estudiante como el espacio entre el nivel de desarrollo actual y el nivel de desarrollo potencial con el apoyo de un experto. Los esfuerzos de la enseñanza deben enfocarse en esta zona [6] y [16].

2.3.3.4. Bloom

Para Bloom, la conducta se compone de tres dominios, el cognoscitivo, el afectivo y el psicomotor. En el dominio cognoscitivo se puede hacer una clasificación en seis niveles de actividad intelectual. Esto se conoce como taxonomía o jerarquía de Bloom [31] [8].

1. Conocimiento.- Implica el proceso de memorización. El estudiante repite la información tal como se la presentó. Comprende el conocimiento de datos y hechos específicos, terminología, convenciones, secuencias, categorías, metodologías, criterios, principios, teorías, etc.
2. Comprensión.- Incluye los procesos para entender e interpreta el mensaje literal de una comunicación. Comprende habilidades de traducción, explicación, extrapolación.
3. Aplicación.- Abarca los procesos caracterizados por la transferencia (generalización) del conocimiento a situaciones parecidas, es decir por la habilidad para llevar a la práctica el conocimiento adquirido.
4. Análisis.- Se refiere a los procesos en que la información recibida se fracciona en sus elementos constitutivos, de modo que se expresen explícitamente la organización, las relaciones y las jerarquías entre las ideas.
5. Síntesis.- Comprende los procesos en que se fusionan varios elementos de tal manera que constituyan un todo que antes no estaba presente, como en el desarrollo de una comunicación original, en la planeación y en la deducción de relaciones abstractas.
6. Evaluación.- Los procesos que implican juzgar el grado de satisfacción de criterios específicos, a partir de evidencia tanto interna como externa.

2.3.3.5. Dubinsky

Desarrolló una metodología para enseñar las matemáticas fundamentada en las teorías de Piaget, basada en el principio de tener vivencias para tener aprendizaje. Las vivencias pueden ser tanto físicas como intelectuales.

Su ciclo metodológico busca estimular la abstracción reflexiva, la herramienta intelectual que impulsa al proceso mental, a través de actividades, clase teórica y ejercicios (por lo que se denomina ACE).

Dubinsky utilizó un lenguaje de programación matemático (ISETL) para generar un complejo ambiente de intercambio entre conocimiento, profesor, alumnos, problemas y computadoras, que fomente la abstracción reflexiva [55].

2.3.3.6. Bruner

Jerome Bruner creó un modelo de la mente humana en el que hay tres distintas mentalidades que compiten por el control:

1. Inactiva.- Hacer, manipular.
2. Icónica.- Reconocer, visualizar, comparar, configurar, concretar.
3. Simbólica.- Abstraer, cadenas de razonamiento.

El aprendizaje debe comenzar en lo concreto y ser llevado a lo abstracto. Alan Kay lo resume en el lema “al hacer con imágenes, se generan símbolos” [57].

Bruner define un paradigma de la transmisión de conocimientos, desde el punto de vista del instructor, usando situaciones estereotipadas que llama *formatos* [16]:

1. Realizar la tarea a enseñar, como ejemplo.
2. Inducir a que el alumno intente lo mismo.
3. Reducir la complejidad del problema, segmentándolo y ayudando.
4. Dominada la tarea, animar a iniciar otra de orden superior.
5. Solamente cuando la tarea es dominada termina la instrucción, es decir, la incorporación del conocimiento adquirido al conocimiento verbalizado.
6. Ahora es posible el discurso entre el maestro y el alumno: ya hay conocimiento compartido.

2.3.3.7. Papert

Partiendo de las ideas de Piaget, Seymour Papert llega a la conclusión de que es más importante ayudar a los niños a aprender como desarrollar y depurar sus teorías, que enseñarles las teorías que se consideran correctas. Con este fin, Papert supervisó la creación del lenguaje de programación LOGO, que aprovecha los requerimientos epistemológicos de las representaciones computacionales (programas) para ayudar a los estudiante formular conceptos [104].

2.3.3.8. Minsky

Marvin Minsky distingue tres tipos de conocimiento, agregando un tercer tipo da la clásica clasificación epistemológica:

1. Conocimiento declarativo
2. Conocimiento procedural (*procedural*)
3. Conocimiento de depuración (*debugging*)

El conocimiento de depuración es el que permite saber qué hacer cuando no se conoce algo, lo que se conoce no funciona, o hay una equivocación. Este conocimiento puede llega a entenderse como el resultado de “aprender a aprender” [73].

2.3.3.9. Anderson

Para John Anderson, cualquier conocimiento se puede representar bajo la forma de reglas de producción [5]. En este sentido, propone la teoría ACT de adquisición de habilidades, basada en:

1. Diferenciar.- El primer paso es distinguir el conocimiento declarativo (datos, hechos) del conocimiento procedural (aplicar los datos conocidos). El conocimiento declarativo es aprendido de la observación e instrucción; la habilidad cognoscitiva radica en convertirlo a conocimiento procedural.
2. Compilar.- El conocimiento procedural se adquiere únicamente al utilizar conocimiento declarativo en un contexto de resolución de problemas, en un proceso denominado “compilación de conocimiento”.
3. Practicar.- Ambos tipos de conocimiento se fortalecen con la práctica. Después de adquirir un conocimiento, practicarlo genera un desempeño más fluido, rápido y seguro.

2.3.3.10. Guilford

Para J. P. Guilford el estudio de la inteligencia va más allá de solamente el coeficiente intelectual (IQ) y desarrolla un modelo de la inteligencia humana en la que combina simultáneamente la información y la inteligencia misma para generar un paradigma que permita desarrollar el funcionamiento mental [11]. Él plantea que la inteligencia no opera sobre lo abstracto sino sobre datos así que habla sobre contenidos y productos de la información además de los procesos y las habilidades intelectuales que operan sobre ellos.

El modelo de la inteligencia de Guilford contiene tres ejes fundamentales:

1. Los Contenidos.- toda información necesariamente tiene contenidos específicos que diferenciará la estructura intrínseca de los datos que posteriormente procesará la inteligencia. Se divide en:
 - a. Figurativo (codificado como F)
 - b. Simbólico (codificado como S)
 - c. Semántico (codificado como M)
 - d. Conductual (C)
2. Los Productos.- Se refieren ala estructura misma de la información y a su construcción mental. Hay seis tipos diferentes de productos:
 - a. Unidades (U)
 - b. Clases (C)
 - c. Relaciones (R)
 - d. Sistemas (S)
 - e. Transformaciones (T)
 - f. Implicaciones (I)
3. Las Operaciones.- los procesos intelectuales manejan la información en un flujo secuencial que es independiente; cada paso condiciona la calidad del siguiente
 - a. Captación (C)

- b. Memoria (M)
- c. Evaluación (E)
- d. Producción Convergente (N)
- e. Producción Divergente (D)

Con este modelo, propone que la inteligencia (y las habilidades asociadas) combina los contenidos de la información para generar productos a través de operaciones mentales. Las habilidades intelectuales son combinaciones formadas por un proceso mental, un producto de información y un contenido. Por ejemplo, un estudio que realizó identifica las habilidades mentales que condicionan la aritmética básica y estas son:

- Captación de Sistemas Simbólicos (CSS)
- Evaluación de Clases Simbólicas (ECS)
- Evaluación de Sistemas Simbólicos (ESS)
- Memoria de Unidades Simbólicas (MUS)
- Memoria de Sistemas Simbólicos (MSS)

Las de las matemáticas son:

- Captación de Sistemas Figurativos (CSF)
- Captación Transformaciones Figurativas (CTF)
- Captación de Relaciones Simbólicas (CRS)
- Memoria de implicaciones Simbólicas (MIS)
- Producción Convergente de Sistemas Simbólicos (NSS)
- Producción Convergente de Implicaciones Simbólicas (NIS)

2.3.3.11. Gardner

Es conocido fundamentalmente por su teoría de las ‘inteligencias múltiples’, que señala que no existe una inteligencia única en el ser humano, sino una diversidad de inteligencias que marcan las potencialidades y acentos significativos de cada individuo, trazados por las fortalezas y debilidades en toda una serie de escenarios de expansión de la inteligencia. En su libro “La Inteligencia Múltiple” propone siete inteligencias a las que posteriormente agregó una octava. Estas son:

1. Lingüística.
2. Lógica-matemática.
3. Corporal y kinésica.
4. Visual y espacial.
5. Musical.
6. Interpersonal.
7. Intrapersonal.
8. Naturalista (agregada después)

2.3.4. Estrategias de Enseñanza Aprendizaje

Debido a que los métodos de enseñanza-aprendizaje no están vinculados con algún paradigma particular se enlistan sin esta clasificación y los más relevantes son los siguientes: Método inductivo (tradicional), Método de la disección (leer primero programas escritos para averiguar que hacen), Método deductivo (utiliza la disección pero además de la lectura también incluye la escritura directa de código), Acercamiento progresivo (ejemplo del robot Karel [79]), Modelo constructivista (Vigotsky), Todos comparten el objetivo de desarrollar la habilidad de la programación en los estudiantes, sin embargo, las investigaciones en el área de conocimiento de las ciencias sociales no concluyen cual es el modelo más efectivo. De hecho, Gardner ha explicado a través de su teoría de las inteligencias múltiples que las personas no tienen una sola inteligencia cognitiva sino que tienen varias que se entremezclan y complementan para que pueda enfrentar de mejor forma los problemas que trata de resolver. Adicionalmente, también existe la teoría de los diversos estilos de aprendizaje que señala que hay personas que aprenden mejor observando (denominadas visuales), otras escuchando (auditivas) y otras a través de la acción y movimiento (llamadas kinestésicas). Todo esto es rematado con la idea humanista de Carl Rogers quien indica que el aprendizaje es un viaje estrictamente individual basado en las experiencias propias y exclusivas de la persona que aprende por lo que tratar de estandarizar en un único método los procesos de aprendizaje resulta infértil. En conclusión, lo que es claro es que existe la necesidad de crear una masa crítica de profesionales competentes en la programación pero no hay una respuesta contundente de cómo debe lograrse esto.

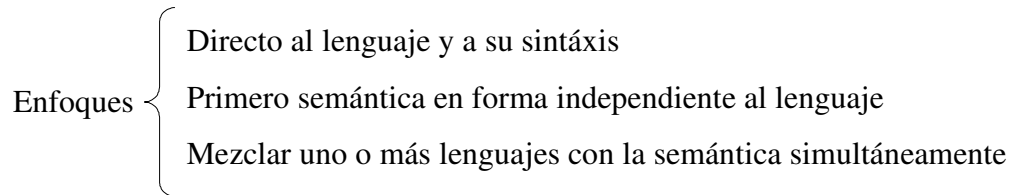
2.3.5. Tácticas de Enseñanza Aprendizaje

Además de los modelos de enseñanza-aprendizaje también existen tácticas sobre como introducir a las personas a la programación. Las que se han detectado con mayor frecuencia son las siguientes: Modelo basado en problemas (PBL por sus siglas en inglés), Modelo basado en casos (CBL), Modelo orientado a proyectos (POL), Modelo basado en aprendizaje colaborativo y Modelo Expositivo. Por supuesto existe una larga lista de tácticas de aula pero las citadas están específicamente orientadas a desarrollar habilidades.

Una habilidad (algunos modelos prefieren el término competencia) no es un conocimiento, es una destreza particular para realizar una tarea. De hecho, se complementa con el conocimiento. La programación es una habilidad mental superior (o competencia) que implica el desarrollo de una serie de habilidades menores que están escalonadas en una especie de taxonomía donde hay habilidades primarias (como el manejo de notación algorítmica, conocimiento de tablas de verdad, etc.) en la parte baja de la taxonomía, hasta habilidades superiores (como la recursividad, manejo de memoria dinámica, etc.) en la parte alta de la misma. Hay autores del área educativa como Bloom (con su taxonomía con seis categorías), Gilford (con su cuadro de categorías) y Gardner (con el modelo de inteligencias múltiples) que proporcionan un marco de referencia para crear una taxonomía sobre este particular y determinar cuales son las habilidades específicas en el arte-ciencia de la programación y que orden pudiera ser el recomendado. Este es un trabajo que debe ser desarrollado por los ingenieros de software en un futuro a muy corto plazo.

2.3.6. Enfoques

Complementando las estrategias y las tácticas se encuentran los enfoques o visiones que tienen los responsables del aprendizaje de los nuevos ingenieros. Estos enfoques se han clasificado en tres básicos:



El enfoque directo al lenguaje sostiene que la mejor forma de aprender a programar es programando en un lenguaje cualquiera y conocer directamente la sintaxis de ese lenguaje y las herramientas que lo soportan. Entre los que apoyan esta idea se encuentran Kernighan y Ritchie [59] y Decker y Hirshfield [33].

El enfoque de conocer primero la semántica va más por la línea de desarrollar la lógica de programación a través del dominio de las estructuras de control representadas por medio de diversas técnicas manteniendo una distancia prudente con la sintaxis de algún lenguaje particular. Finalmente, por supuesto que es necesario aterrizar en algún lenguaje pero sostienen que si los programadores desarrollan habilidades en el diseño de soluciones algorítmicas podrán cómodamente moverse entre diversos lenguajes. Además, señalan que uno de los principales problemas de iniciar directo con algún lenguaje es que rápidamente se cae en el problema de estudiar la interfaz de programación (IDE) y las características de la herramienta perdiendo de vista el tema central que es la capacidad de resolver problemas en lugar de cómo utilizar un editor. Entre los autores que se proponen a favor de este enfoque se encuentran Budd [17], Alcalde y García [3], Joyanes [55], Gosling [47], Voss [102], Patis [79] y Dijkstra quien para escribir su libro en [35] justifica su decisión de no elegir un lenguaje específico de la siguiente forma: “Cuando se comienza un libro como este, uno se enfrenta de inmediato a la pregunta: ‘Que lenguaje de programación voy a utilizar?’. ¡Esto no es únicamente una cuestión de presentación! El más importante, pero también más elusivo, aspecto de cualquier herramienta es la influencia que produce en los hábitos de aquellos que tratan de entrenarse en su uso. Si la herramienta es un lenguaje de programación, su influencia – nos guste o no – es directa en nuestros hábitos de pensar. Después de haber analizado esta influencia con mi mayor empeño, he llegado a la conclusión de que ninguno de los lenguajes de programación existentes, ni ningún subconjunto de los mismos coincidirían con mi propósito”

El enfoque de mezclar semántica con sintaxis de lenguajes sostiene que es tan importante dominar la semántica de las estructuras de control como la sintaxis de un lenguaje y que debido a la necesidad de programar se recomienda utilizar algún lenguaje en específico cuidando de concentrarse en los temas relevantes y no en los temas accesorios. Glenn [45] es uno de los que proponen esta visión.

Cada una de estas visiones tiene sus pros y sus contras. Todos los autores reconocen que el desarrollar las habilidades de programación en algún lenguaje es el fin último pero que esta habilidad depende a su vez de la habilidad de solucionar problemas por medio de diseño de

soluciones algorítmicas que son, por su propia naturaleza, independientes del lenguaje. Sin embargo, son más los autores que se manifiestan a favor de privilegiar las habilidades de diseño algorítmico a través del fortalecimiento del dominio de la semántica de cada una de las estructuras de control y que el dominio del lenguaje puede darse posteriormente.

2.4 Herramientas Visuales de Programación

En esta sección se describen varios ambientes visuales para programar (LVP) pero también se incluyen sistemas que utilizan representaciones gráficas de un programa textual.

2.4.1. MacGnome (1987)

MacGnome es un editor de estructuras, es decir, un ambiente de programación que opera directamente en el árbol sintáctico del programa, a diferencia de los ambientes tradicionales que generan el árbol hasta el momento de compilar. Un cambio al programa consiste en el reemplazo de un hueco (placeholder) por un sustituto legal. En realidad, cada cambio consiste en la transformación de un programa sintácticamente correcto a otro.

MacGnome pone un énfasis especial en la usabilidad del sistema. La interfaz facilita la navegación, la modificación y la ejecución del programa. Para fomentar el desarrollo de habilidades de alto nivel, como planeación, abstracción y visualización, cuenta con diversas vistas del programa. Por ejemplo, vista de esqueletos de subprogramas, vistas gráfica de la estructura del programa o mirador (match) para estructuras complejas de datos [72].

Por ejemplo, la Figura 28 muestra tres de las representaciones de MacGnome para un programa. En una ventana se puede ver la totalidad del código, en otra se muestran las declaraciones de las subrutinas y en la restante una subrutina particular.

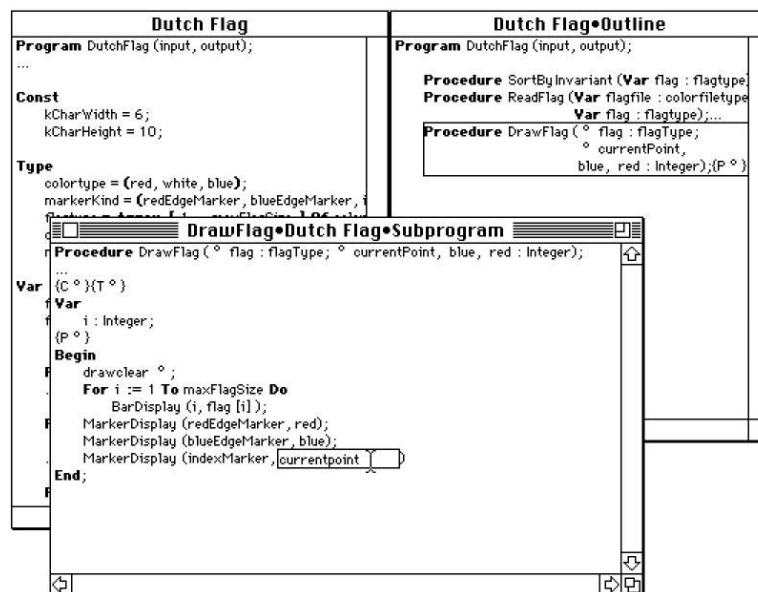


Figura 28. Ejemplo de MacGnome

2.4.2. LabView (1990)

LabView es un LVP basado en los diagramas de circuitos electrónicos y en el modelo de flujo de datos. Los operadores se representan como compuertas, y cualquier programa puede ser abstraído como un nuevo operador (subprograma). Las estructuras de control se expresan por encerramiento espacial, a través marcos con valores de control. Para expresar condicionales se definen subprogramas alternativos dependientes del valor de control. Toda la entrada/salida al programa se representa gráficamente en un tablero de control [49].

A diferencia de la mayoría de los LVP que son prototipos de investigación, LabView es un producto comercial relativamente exitoso.

La Figura 29 muestra un pequeño programa en LabView. El marco gris representa un ciclo, que se repetirá mientras que el *switch* se encuentre en estado *On*. Cada iteración se lee la temperatura de un “termómetro” y se envía a una gráfica.

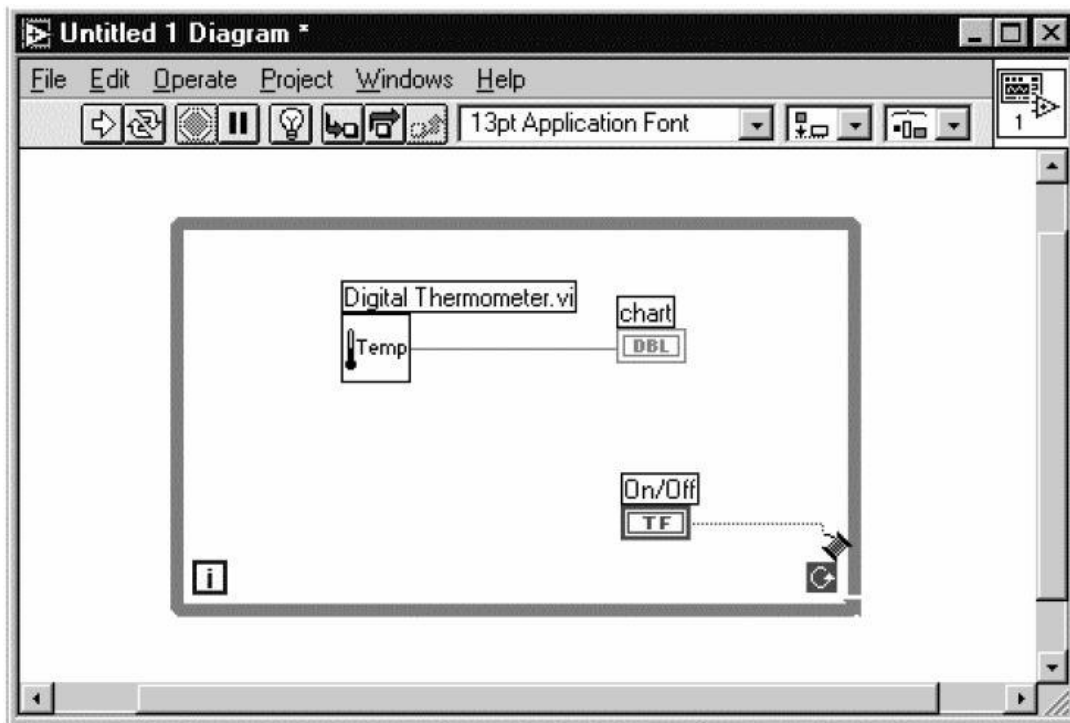


Figura 29. Ejemplo de LabView

2.4.3. Prograph (1992)

Prograph es un LVP orientado a flujo de datos y a objetos. El lenguaje se construye alrededor de métodos representados por iconos, que se conectan a través de líneas que transportan objetos. Un método puede tener varios casos, que se eligen a través de un condicional. Cada caso se representa en una ventana independiente, lo que puede generar un árbol muy profundo de sub-ventanas. En una ventan los objetos fluyen de una barra superior de entrada, hacia abajo, hasta llegar a una barra de salida. Cada método se representa con un icono que acepta objetos encima y produce objetos abajo. [49] [50].

La Figura 30 muestra un programa en Prograph que, aplicando el teorema de Pitágoras despliega la hipotenusa de un triángulo con catetos de longitud 3 y 4.

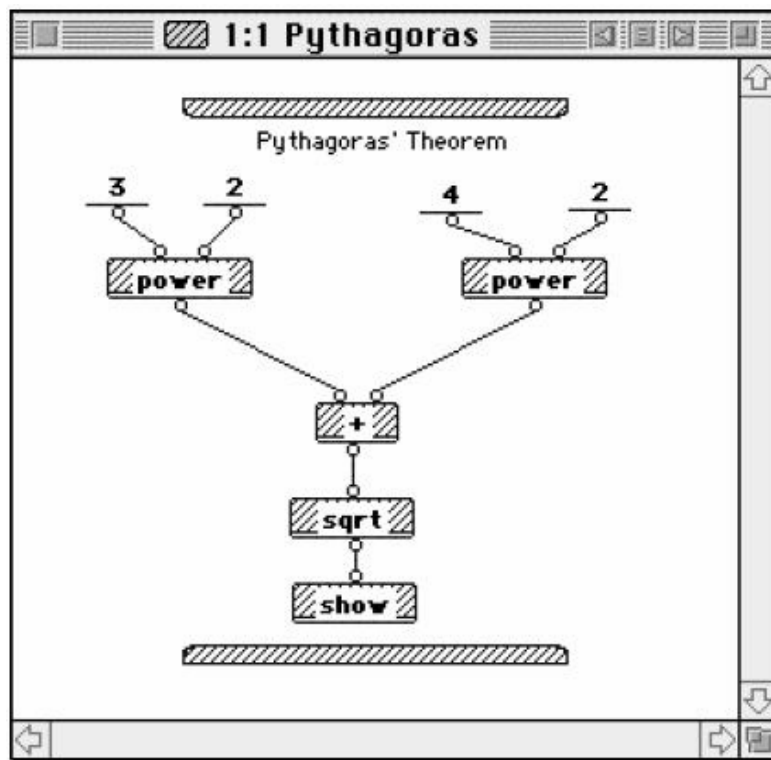


Figura 30. Ejemplo de Prograph

2.4.4. Baccii (1994)

BACII es un ambiente icónico que va guiando la programación del usuario a través de las distintas partes de un programa en Pascal. El usuario necesita seguir un orden determinado; por ejemplo, lo primero que tiene que hacer es determinar el nombre del programa y declarar una variable. La mayor parte de la interacción es a través de la selección de iconos (que representan palabras reservadas en Pascal), y configurándolos a través de ventanas de diálogo. La estructura del programa es una variante de un diagrama de flujo tradicional, que reemplaza las figuras tradicionales por iconos. BACII puede generar código en Pascal, pero no puede ejecutar los diagramas; en este sentido, no es un LVP propiamente [20].

Por ejemplo, la Figura 31 muestra un programa en construcción. En la parte superior hay una estructura IF, con huecos para un enunciado normal y uno compuesto. Va seguida de una estructura CASE con 4 casos. Finalmente hay un ciclo WHILE-DO, con un hueco para un enunciado compuesto. Del lado izquierdo se puede ver la caja de herramientas con todas las estructuras y enunciados que se pueden colocar en el programa.

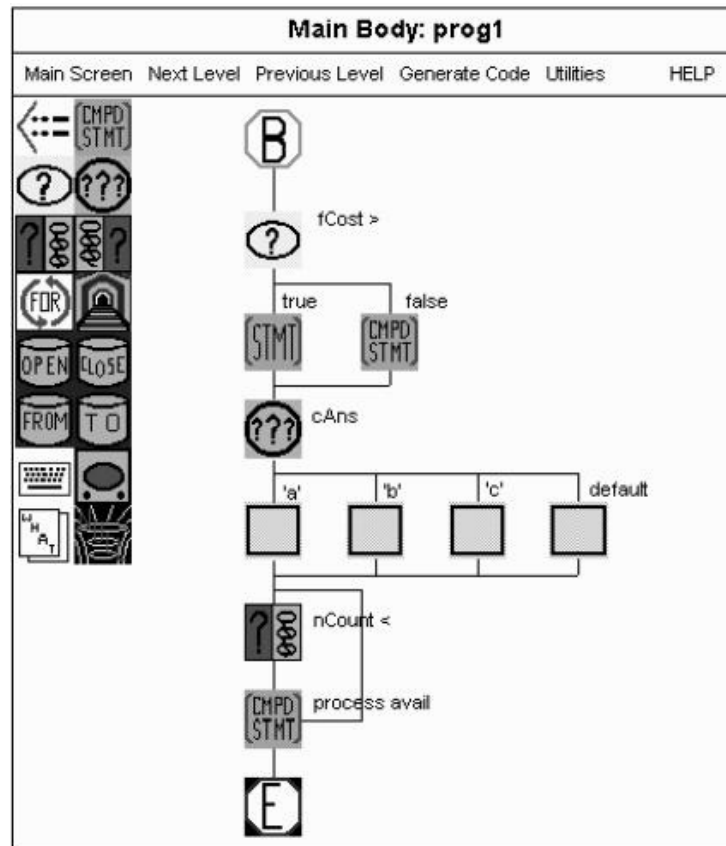


Figura 31. Ejemplo de BACII

2.4.5. KidSim (1994)

KidSim es un LPV basado en reglas visuales, diseñado para que los niños puedan crear simulaciones gráficas. Un programa se compone de reglas y actores en un mundo. A través de demostración se crean reglas de reescritura gráfica, que luego pueden ser editadas visualmente. El mundo de KidSim es una malla de casillas que pueden ocuparse por distintos objetos gráficos; las reglas operan en patrones de ocupación de casillas [100].

La Figura 32 muestra un ejemplo de un programa de KidSim. En la parte superior se puede ver el mundo en el estado inicial. Se puede ver que los actores son dos tipos distintos de niño, botellas y un basurero. En la parte inferior se ven todas las reglas visuales de producción. Al ejecutar este programa el niño de la derecha va por una botella y se la pasa al compañero de la izquierda. Este bebe de la botella y luego la sigue pasando. Finalmente la botella termina en el basurero. Mientras tanto, el niño de la derecha ha continuado tomando y pasando las botellas.

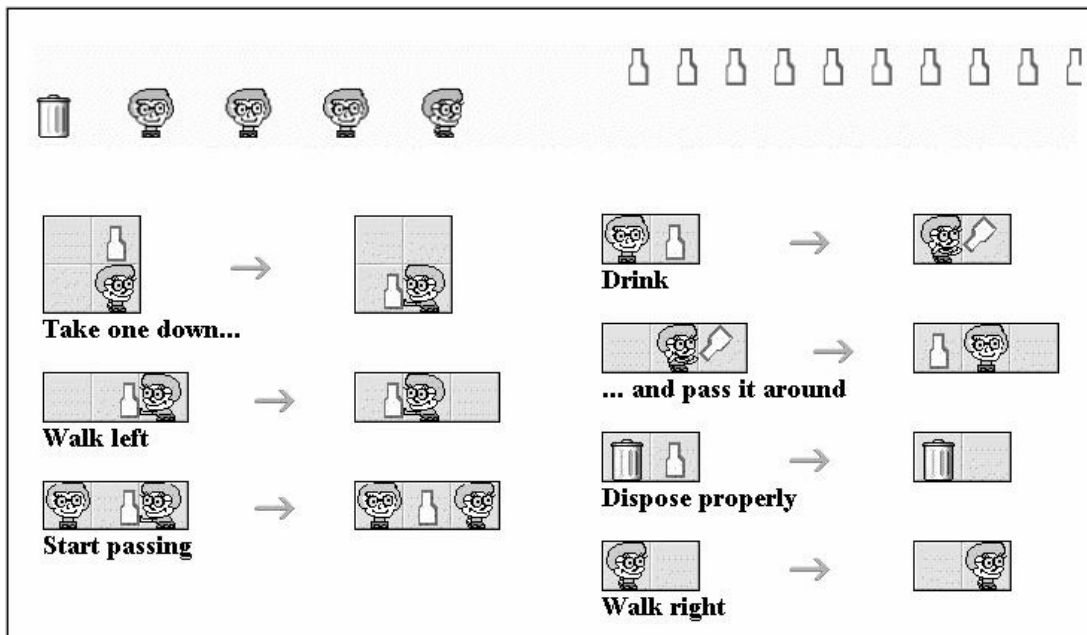


Figura 32. Ejemplo de KidSim

2.4.6. FlowCoder (1995)

FlowCoder es una herramienta de ingeniería en reversa, que permite expresar en forma de diagrama el código de una variedad de lenguajes. Esto lo hace a partir de traductores bidireccionales, uno para cada lenguaje específico. Cuenta también con simuladores que permiten ejecutar y depurar directamente los diagramas.

Los diagramas están basados en el modelo de flujo de control. Cada línea de texto del programa en el lenguaje original está presente, lo que es terriblemente redundante. Esto también implica que un programa en FlowCoder/Pascal no puede ser traducido a otro lenguaje, y que es necesario conocer tanto FlowCoder como Pascal para programarlo.

FlowCoder es un producto comercial (<http://flowlynx.com>), y puede ser personalizado e integrado a ambientes profesionales de programación.

La Figura 33 muestra un ejemplo de código en FlowCoder, en su versión de pascal. Dentro de un ciclo REPEAT-UNTIL se encuentran dos ciclos WHILE-DO y una estructura IF. Se puede ver que se incluye todo el texto del programa en Pascal, incluyendo redundantemente construcciones como BEGIN y THEN. También es notable que los dos tipos de ciclos tengan una misma representación gráfica.

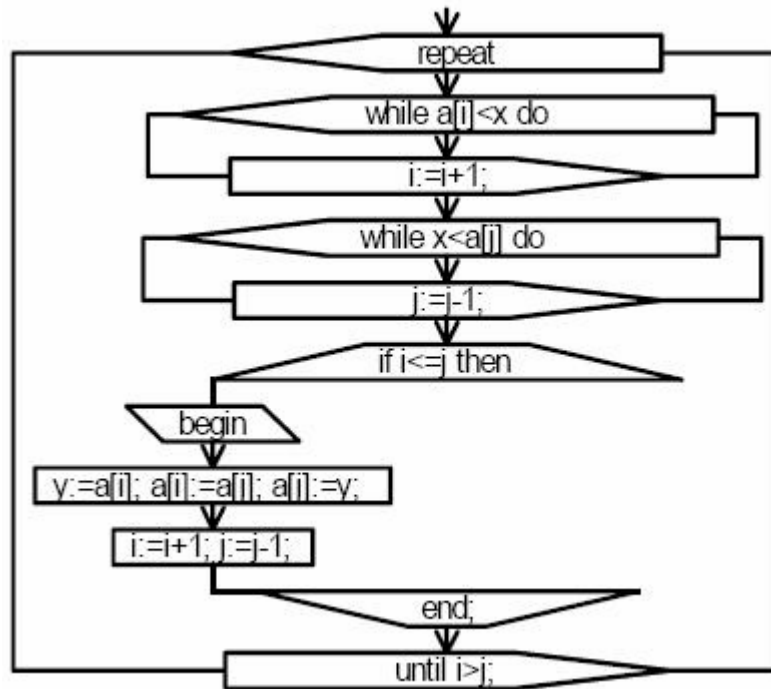


Figura 33. Ejemplo de FlowCoder

2.4.7. ToonTalk (1995)

ToonTalk es un LVP basado en el paradigma de restricciones concurrentes, que es una variante orientada a objetos de programación lógica. Las construcciones de este paradigma están mapeadas a objetos gráficos. Por ejemplo, los métodos se representan como robots y los objetos como casas. El usuario programa por demostración, y puede después generalizar su programa eliminando constantes manualmente. El ambiente está permanentemente animado, borrando la diferencia entre tiempos de diseño y ejecución [100].

2.4.8. Ambientes Visuales de Programación

En general, programar en estos ambientes no es sencillo, tanto por la complejidad de los lenguajes como por la dificultad de edición. La tabla 5 se compara las características de estos sistemas. Es de subrayar que los ambientes de flujo de control no logran reducir significativamente la complejidad de sus contrapartes textuales.

2.5 Reflexiones

Diversas son las reflexiones que se pueden hacer tras una revisión histórica como la expuesta en cada uno de los dominios analizados. A continuación se relacionan las más importantes:

- ✓ La realidad actual en materia de diseño algorítmico es resultado del proceso histórico que se ha vivido.
- ✓ La programación aún requiere de habilidades personales que rayan en la capacidad artística para el diseño de algoritmos.

Sistema	MacGnome	LabView	ProGraph	BACII	FlowCoder	KidSim / ToonTalk
Medidas						
Paradigma	Flujo de control	Flujo de datos	Flujo de datos	Flujo de control	Flujo de control	Reglas visuales de producción
Visual	Como apoyo	Sí	Si	Sí	Sí	Sí
Lenguaje	Pascal	LabView	ProGraph	Pascal	Varios	KidSim/ Toon Talk
Ejecutable	Sí	Sí	Si	No	Sí	Sí
Complejidad	Medida	Alta	Alta	Alta	Muy alta	Baja
Usabilidad	Regular	Mala	Mala	Mala	Muy mala	Regular

Tabla 5. Características de los sistemas visuales

- ✓ La calidad del software es un tema prioritario y de preocupación mundial
- ✓ La calidad de cualquier producto de software depende del talento humano y de lo que éste ha aprendido dentro del marco de su propio desarrollo.
- ✓ El talento humano es posible desarrollarlo a partir de combinar una colección de conocimientos, habilidades y destrezas así como la metodología con que se enseñan y aprenden.
- ✓ La correcta aplicación del teorema de la estructura de Jacopini sigue vigente a pesar del paso de los años y de los paradigmas y hay ausencia de actualidad sobre esto en la POO y modelos posteriores.
- ✓ Los métodos y herramientas de diseño de algoritmos han caído en desuso con excepción del pseudocódigo.
- ✓ Los principios de refinamiento y descomposición del problema en partes e identificación de entradas, salidas, restricciones y procesos siguen vigentes, sin embargo, no con la intención de transitar del dominio del problema al dominio de la solución, pero si para el diseño algorítmico y no son explotados a todo su potencial ni se han desarrollado nuevos métodos para su ejecución.
- ✓ La didáctica de la programación y la forma como aprenden los futuros ingenieros no puede ser responsabilidad exclusiva de los pedagogos, debe ser co-responsabilidad de los científicos e investigadores de las ciencias de la computación.
- ✓ Si bien desde el punto de vista cognoscitivista están muy claros los tópicos que deben ser del dominio de un buen programador, la identificación científica de las habilidades mentales que requiere y la forma de desarrollarlas es un tema que aún no ha sido investigado.
- ✓ Ya se han realizado esfuerzos por la creación de herramientas visuales de programación específicamente para modelar algoritmos pero éstos presentan constantes problemas de visibilidad, navegabilidad y modificabilidad ocasionando que su uso sea abandonado y las personas prefieran trabajar directamente en los lenguajes 3GL.

Capítulo 3. Modelo para la Construcción de Algoritmos Apoyados en Heurísticas

3.1 Introducción

La construcción de algoritmos es una tarea compleja cuyo resultado se manifiesta a través del arte-ciencia de la programación. Sleeman [95] explicó que cuando un programador se enfrenta al reto de crear un programa en realidad tiene que resolver varios problemas. El primero es descomponer la tarea en un plan detallado y realizable que justamente solucione el problema; el segundo corresponde a la implementación de este plan en un lenguaje de programación; y finalmente el tercero es depurar el programa resultante lo cual puede ser tan complejo como los dos anteriores.

Por su parte, Soloway [96] afirma que el programa en si mismo no debe ser únicamente un mecanismo que le indique a la computadora como debe resolver un problema, sino también una explicación que le dice, al propio programador, la forma en que éste se resuelve. Por ello, aprender a programar consiste en aprender a construir mecanismos y explicaciones. Esto quiere decir que la escritura de un algoritmo implica que su autor debe poder resolver el problema y entender como llegar a esta solución antes de que el programa pueda nacer; esto trasciende el campo de la programación.

Tradicionalmente, la enseñanza de la programación se ha concebido como el proceso a través del cual un estudiante conoce y aplica la sintaxis y semántica de un lenguaje de computadora. Incluso, como se explicó en el capítulo 2, hay discusiones sobre que debería de ser primero; hay quienes defienden el enfoque directo al lenguaje (sintaxis primero) y otros prefieren desarrollar primero la lógica independiente al lenguaje (primero semántica). Sin embargo, las investigaciones de Sleeman y de Soloway han demostrado que el verdadero problema que tienen los principiantes radica en el proceso de ensamblar las piezas. Los programadores expertos conocen mucho más que sintaxis y semántica; saben como resolver una variedad de problemas articulando teorías, principios, métodos y soluciones.

Partiendo del principio de que el aprendizaje de la programación trasciende las fronteras mismas de la programación y se interna en los procesos mentales de solución de problemas se entra al campo de la psicología educativa.

500 años antes de Cristo, Tales de Mileto creó un modelo del universo geocéntrico en el cual la tierra era plana y flotaba sobre agua. Esta cosmovisión se utilizó durante prácticamente dos mil años antes que Copérnico y Galileo se atrevieran a cuestionar su veracidad. De igual forma, en la Grecia antigua se fundaron, en los jardines de Academus, las primeras “Academias” dando pie al nacimiento de las escuelas. También han pasado más de dos mil años antes de que se cuestionara la forma como se educa a las personas. La segunda mitad del siglo pasado ha sido rica en nuevas ideas sobre las formas de educación transitando de modelos inductivos-receptivos a modelos intuitivos-activos. Autores como Piaget, Vigotsky, Bloom, Rogers, Bruner y Guilford, entre otros, han aportado interesantes conceptos sobre la enseñanza y sobre el aprendizaje.

Particularmente Dubinsky desarrolló una metodología para enseñar matemáticas utilizando las teorías de Piaget. Sin embargo, no son los pedagogos ni los psicólogos educativos los que propondrán soluciones al problema del aprendizaje de la programación.

Adicional al enfoque de la teoría de la programación y de las corrientes educativas, se encuentra la parte instrumental. Esta es una pieza irrenunciable en el proceso de construcción de algoritmos. Las habilidades desarrolladas por una persona sobre las teorías para resolver problemas se revelan a través del uso de las herramientas propias de la profesión.

Muchas y diversas son las herramientas actuales para construir código ejecutable en una computadora y a pesar de considerarlas imprescindibles, ninguna de ellas podrá sustituir la habilidad creativa del Ingeniero.

Ni el estudio de la teoría de la programación, ni el de las teorías del aprendizaje, ni el desarrollo de nuevas herramientas lograrán en lo individual solucionar el problema de resolver problemas a través de la construcción de algoritmos. Cada una de estas tres dimensiones (teorías, habilidades y herramientas) son interdependientes en un mundo complejo⁴ donde los enfoques positivistas de causa y efecto han sido sobrepasados. Hay múltiples causas y múltiples efectos, todo está conectado.

Para superar las limitaciones que el tratado individual y separado de las tres dimensiones implicaría se propone un modelo holístico que integre la triada de la programación explicada en el capítulo 1. Así, el modelo de Construcción de Algoritmos Apoyado en Heurísticas contempla un conjunto de elementos para el modelo teórico, otro para el área de habilidades y uno más para el tema instrumental. En este trabajo, el conocimiento holístico⁵ se entiende como el estudio simultáneo de todos los aspectos que forman una cosa y la manera en que todas esas partes interactúan entre sí para dar como resultante el objeto completo.

El modelo de Construcción de Algoritmos Apoyado en Heurísticas aspira a ser un modelo holístico en el sentido epistemológico donde el estudio y comprensión de las partes y sus relaciones es importante pero indivisible del todo que forman.

3.2 Descripción del Modelo

Como se puede apreciar en la figura 34, el modelo está formado por tres dimensiones:

Marco Teórico.- Formado por las teorías, paradigmas, procedimientos y metodologías que conforman un marco conceptual sobre el cual se modela el dominio del problema y el dominio de la solución. El componente principal del modelo es una metodología de Microingeniería de Software.

⁴ El concepto de complejo no tiene relación con el de dificultad. El término complejo se refiere al número de relaciones, dependencias e interdependencia cruzadas entre los componentes del sistema.

⁵ Holístico proviene del griego *holos* (όλος) que significa todo. Es el estudio del todo, relacionándolo con sus partes pero sin separarlo del todo.

Talento Humano.- Que representa las habilidades individuales que se poseen, y en particular las habilidades de programación. El componente principal de este dominio es un modelo de habilidades de programación.

Herramientas de Desarrollo.- Formadas por distintos componentes de hardware y software que permiten construir, implementar y operar la solución. El principal componente de este dominio es un lenguaje de modelado incrustado dentro de una arquitectura para el Diseño de Algoritmos Asistido por Computadora. (DAAC).

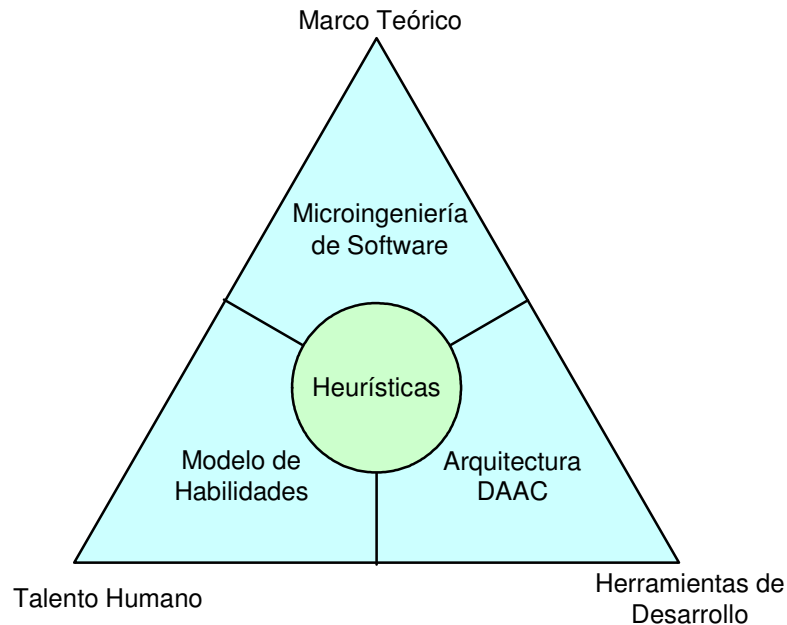


Figura 34. Modelo para la construcción de algoritmos apoyado en heurísticas

Cada una de estas dimensiones tiene sus propios componentes y sus respectivas relaciones e interdependencias. En las siguientes secciones se explicará cada una de ellas.

3.3 Heurísticas

Se denomina heurística a la capacidad de un sistema para realizar de forma inmediata innovaciones positivas para sus fines. La capacidad heurística es un rasgo característico de los humanos, desde cuyo punto de vista puede describirse como el arte y la ciencia del descubrimiento y de la invención o de resolver problemas mediante la creatividad y el pensamiento lateral o pensamiento divergente.

La etimología de heurística se presentó en el capítulo uno, dónde se explica que proviene del griego *heurísto* (*εὐρίστω*) y significa inventar. Este término ha sido utilizado en muchas formas gramaticales y por distintas ciencias. Cuando se usa como sustantivo, identifica el arte o la ciencia del descubrimiento, una disciplina susceptible de ser investigada formalmente. Cuando aparece como adjetivo, se refiere a cosas más concretas, como estrategias heurísticas, reglas

heurísticas o silogismos y conclusiones heurísticas. Por supuesto que estos dos usos están íntimamente relacionados ya que la heurística usualmente propone estrategias heurísticas que guían el descubrimiento.

La popularización del concepto se debe al matemático George Polya, [84] explicado en el capítulo dos. Habiendo estudiado tantas pruebas matemáticas desde su juventud, quería saber como los matemáticos llegan a ellas. Su libro “Como resolverlo” [84] contiene la clase de recetas heurísticas que trataba de enseñar a sus alumnos de matemáticas. Cuatro ejemplos extraídos de él ilustran el concepto mejor que ninguna definición:

- Si no consigues entender un problema, dibuja un esquema.
- Si no encuentras la solución, haz como si ya la tuvieras y mira a ver qué puedes deducir de ella (razonando hacia atrás).
- Si el problema es abstracto, prueba a examinar un ejemplo concreto.
- Intenta abordar primero un problema más general (es la “paradoja del inventor”: el propósito más ambicioso es el que tiene más posibilidades de éxito).

En la matemática, la heurística existe desde la Grecia antigua. Sin embargo, la formalización y el alto grado de rigor en matemáticas le han restado importancia al estudio del descubrimiento, considerándolo más bien de interés para la psicología. Aunque existe el campo de la teoría de la demostración, éste nada tiene que ver con encontrar patrones de demostración o reglas para encontrar las demostraciones de los teoremas.

En las ciencias de la computación se ha hecho una acepción de heurística en la ANSI/IEEE Std 100-1984, donde explican que trata de aquellos métodos o algoritmos exploratorios para la resolución de problemas, en los que las soluciones se descubren por la evaluación del progreso logrado en la búsqueda de un resultado final. Se trata de métodos en los que, aunque la exploración se realiza de manera algorítmica, el progreso se logra por la evaluación puramente empírica del resultado. Se gana eficacia, sobre todo en términos de eficiencia computacional, a costa de la precisión. Las técnicas heurísticas son usadas por ejemplo en problemas en los que la complejidad de la solución algorítmica disponible es función exponencial de algún parámetro; cuando el valor de este parámetro crece el problema se vuelve rápidamente inabordable. Las técnicas heurísticas no aseguran soluciones óptimas, sino solamente soluciones válidas, aproximadas; y frecuentemente no es posible justificar en términos estrictamente lógicos la validez del resultado.

En psicología la heurística se relaciona con la creatividad y se ha propuesto que sea aquella regla sencilla y eficiente para orientar la toma de decisiones y para explicar en un plano práctico cómo las personas llegan a un juicio o solucionan un problema. Usualmente una heurística opera cuando un problema es complejo o el problema trae información incompleta. En general, una heurística puede considerarse como un atajo a los procesos mentales activos y, por lo tanto, es una medida que ahorra o conserva recursos mentales. Las heurísticas funcionan efectivamente en la mayoría de las circunstancias pero, sin embargo, también pueden conducir a errores sistemáticos en la toma de decisiones o el desarrollo de juicios. La creación de soluciones heurísticas frecuentemente arranca de un razonamiento por analogía.

Con estas concepciones de heurística se puede decir que una teoría científica tiene un alto valor o componente heurístico si es capaz de generar nuevas ideas o inducir nuevas invenciones sin importar si la teoría es cierta o no.

En el caso de estudio que se aborda se debe considerar el enfoque computacional acompañado del enfoque psicológico ya que el objetivo es la identificación de heurísticas que dirijan el diseño de algoritmos pero desde la perspectiva humana. Las heurísticas propuestas en este trabajo no son para que una computadora, a través de algoritmos que las implementan, llegue a la solución de un problema; más bien son para que las personas puedan progresivamente descubrir el algoritmo que resuelve un problema computable.

3.3.1. Definiciones

En la sección anterior se analizaron varias concepciones de heurísticas desde el punto de vista de las matemáticas, de las ciencias de la computación y de la psicología. Combinando los elementos más relevantes de cada concepción se integra una definición formal para este trabajo:

“Heurística es una regla sencilla y eficiente que pueda orientar en el proceso de diseño de un algoritmo para la resolución de problemas computables, en los que los diseños se descubren por la evaluación del progreso logrado en la búsqueda de un algoritmo final.”

En la sección 3.3.2 se proponen un conjunto de reglas heurísticas, que se han denominado “Reglas de la Lógica” que aspiran a cumplir con esta definición. Ese conjunto de reglas hace uso de ciertos términos que es importante clarificar para su posterior explicación, ejemplificación y aplicación por lo que se presentan las siguientes definiciones:

- **Operación.-** Actividad específica que realiza la computadora por indicaciones del programador. Por ejemplo: la lectura del valor de una variable desde el teclado, la impresión de un resultado en la pantalla, la resolución de un cálculo matemático y la asignación del resultado a una variable, la evaluación de una condición para determinar si es verdadera o falsa, la invocación de un método de un objeto, el control de una repetición, etc. En un lenguaje visual cada operación se representa por un símbolo específico. Ver figura 35.

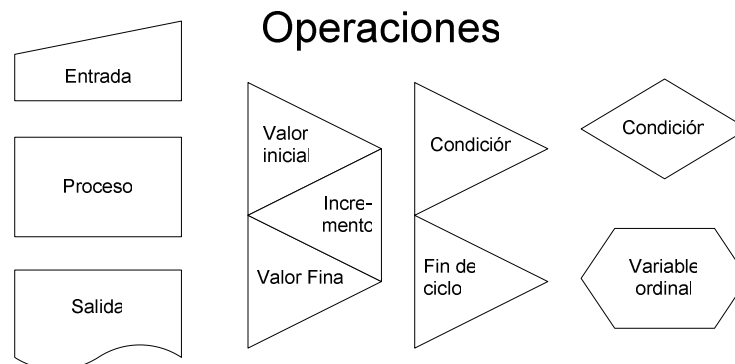


Figura 35. Ejemplos de símbolos que representan operaciones

- Estructura de Control.- Conjunto de operaciones que permiten representar alguna de las tres estructuras indicadas en el teorema de Bohém-Jacopini: secuencia, condición y repetición. Actualmente se identifican tres tipos de condición y tres tipos de repetición haciendo un total de siete estructuras de control: secuencia, if-then, if-then-else, case, for, while y until. Las estructuras de control se consideran unidades de programación indivisibles. En un lenguaje visual se representan por un rectángulo cuyas líneas verticales izquierda y derecha se dibujan con doble línea. Ver figura 36.

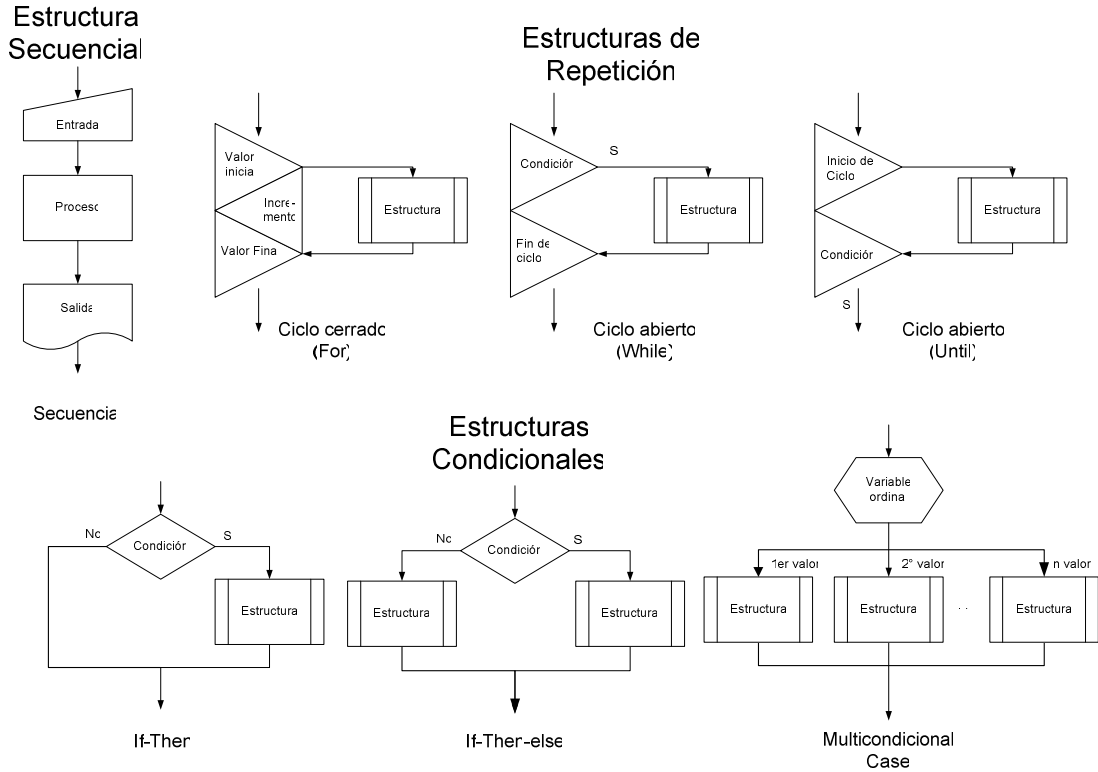


Figura 36. Estructuras de control.

- Tarea.- Conjunto de estructuras de control cuyo inicio y fin están claramente indicadas por elementos terminales. Típicamente representan la solución completa a un problema o acción que debe resolverse o ejecutarse por un equipo de cómputo. Ver figura 37.

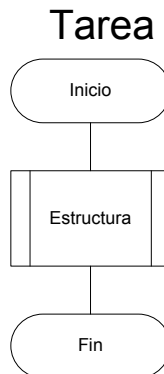


Figura 37. Diagrama de tarea

- Acción.- Es un subconjunto de las operaciones en las que solamente se consideran las de entrada, de proceso y de salida. Ver figura 38.

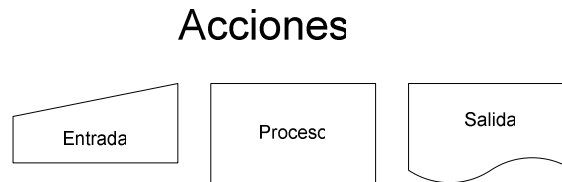


Figura 38. Símbolos de acciones.

- Entrada.- Operación que representa la lectura de uno o más valores desde algún dispositivo para ser almacenado en una o más variables.
- Proceso.- Operación que representa la ejecución de un cálculo y asignación del resultado a una variable. También puede representar la invocación de un método de un objeto o de una función.
- Salida.- Operación que representa visualización del valor de una o más variables en algún dispositivo.
- Procedimiento.- Conjunto de estructuras de control organizadas y secuenciadas de tal forma que muestran la solución parcial de una tarea.

3.3.2. Reglas Heurísticas de Modelado Algorítmico

Las reglas de la lógica tienen que ver con el orden y combinación que deben guardar las estructuras de control para obtener los resultados deseados y son las siguientes:

1. **Regla de Secuencia.**- Todas las tareas requieren de al menos una salida y cero, uno o más procesos y entradas. Primero se ejecutan las operaciones de entrada, luego las de proceso y finalmente las de salida.
2. **Regla de Equivalencia.**- Las acciones (es decir, los procesos de entrada, proceso y salida) se consideran equivalentes entre sí y por tanto toda acción de proceso puede convertirse en entrada o salida y viceversa.
3. **Regla de Composición.**- Un proceso complejo puede dividirse en subprocesos más simples colaborando en la solución y varios procesos individuales pueden combinarse en uno solo más complejo.
4. **Regla de Colaboración.**- La(s) salida(s) de un proceso puede(n) usarse como entrada(s) de otro.
5. **Regla de Cerradura.**- En todo lugar donde haya una acción (es decir, una entrada, un proceso o una salida) puede colocarse una estructura de control. En todo lugar que haya una estructura de control puede colocarse una acción.
6. **Regla de Creación.**- En una tarea o procedimiento solo puede haber nuevas acciones o estructuras. Estas únicamente pueden agregarse entre acción y acción o entre acción y estructura o entre estructura y estructura.
7. **Regla de Integridad.**- Las estructuras de control no pueden cambiar su diseño. Las operaciones son parte integral de las estructuras de control, no son estructuras independientes.

8. **Regla del Flujo.**- La secuencia de ejecución está dictado por el diseño de cada estructura de control. Esta puede ser secuencial o anidada.
9. **Regla de Existencia.**- Todas las variables antes de usarse deben estar definidas y si son parte de un proceso de cálculo deben tener un valor válido.
10. **Regla de Restricción.**- Toda variable que deba cumplir con dominios del mundo real o del mundo virtual debe ser validado a través de una estructura de restricción.
11. **Regla de Repetición.**- Toda Estructura o Acción que se escriben en más de una ocasión pueden ser introducidos con una estructura de repetición.
12. **Regla de Combinación.**- Uno o más procedimientos pueden combinarse para formar otro procedimiento o una tarea. La composición puede darse por concatenación, anidación, fusión o adaptación.

Su ejemplificación y aplicación se explica en la sección 3.4.7. Esto se debe a que su uso se realiza en el contexto del modelo de microingeniería de software mismo que se expone en la siguiente sección.

3.4 Dimensión Teórica: Microingeniería de Software

3.4.1. Microingeniería

La metodología de microingeniería de software aspira a trazar una ruta formal que conduzca de forma reflexiva a una persona del dominio del problema al dominio de la solución. Recibe el nombre de microingeniería porque sus fronteras inician en las definiciones de los contratos que los métodos de las clases deben cumplir (incluyendo pre y post condiciones, así como el respeto del estado de los objetos) y terminan en la implementación de dicho método. Atiende la parte más concreta del proceso total de ingeniería.

Esta metodología complementa las actuales de ingeniería de software donde estas operan en las capas de abstracción del problema más altas y justamente concluyen en la definición de especificación del método de la clase. A partir de ahí, tradicionalmente los programadores inician la codificación del método como principal actividad de la etapa de implementación. Es justamente éste el nicho de aplicación de la microingeniería.

La metodología de microingeniería incluye un ciclo de vida completo para la construcción del algoritmo que, como se puede observar en la figura 39, está formado por cinco etapas: análisis, diseño, construcción, pruebas y liberación. Cada una de ellas debe crear productos específicos y cuenta con herramientas y reglas para lograrlo además de apoyarse en las heurísticas de construcción de algoritmos. Este ciclo de vida está armonizado con los típicos modelos de análisis y diseño e ingeniería de software. Si la microingeniería es utilizada como estrategia de enseñanza aprendizaje en las primeras materias de la carrera (incluso antes de haber estudiado formalmente ingeniería de software) permitirá la introducción de niveles de abstracción más altos de forma más simple gracias a la familiaridad que el estudiante ya tendría con los conceptos y etapas respectivas.

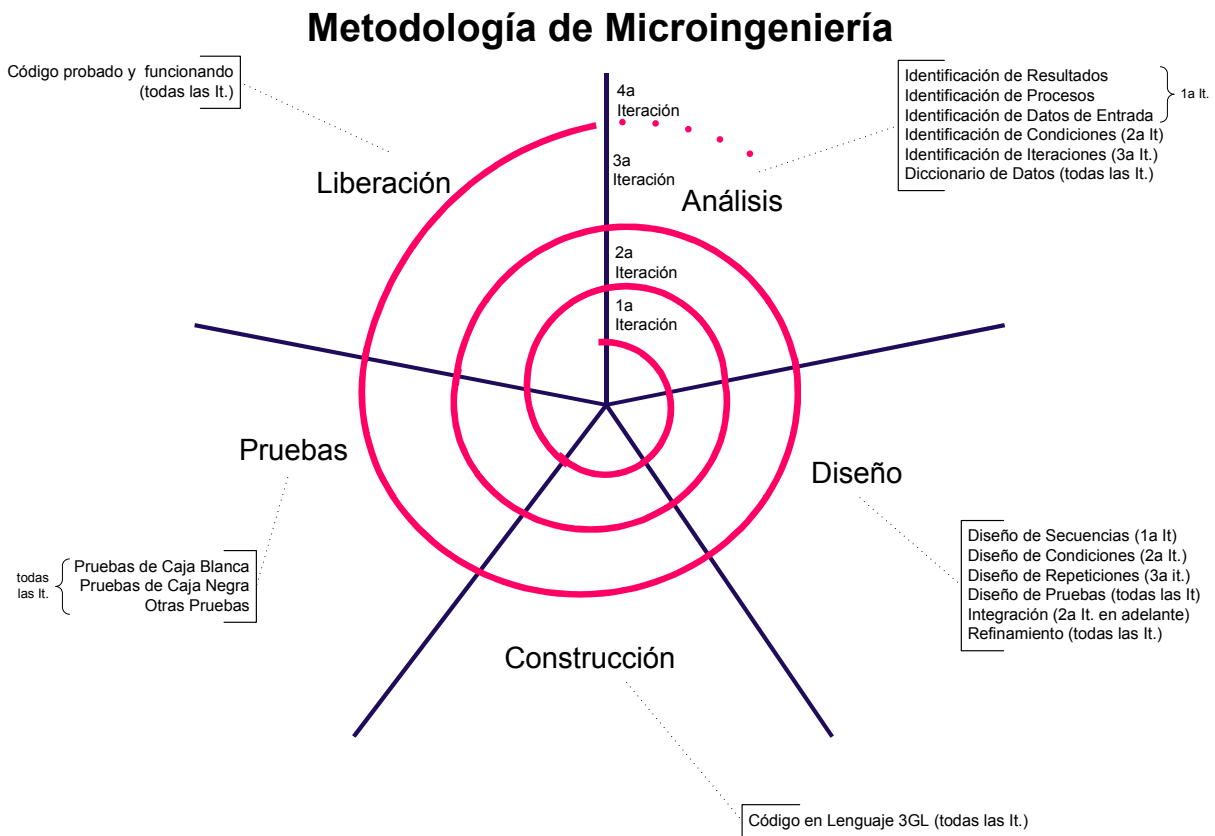


Figura 39. Metodología de microingeniería de software

La metodología está diseñada en forma de espiral con el objetivo de aplicar una estrategia incremental en el proceso de diseño y construcción del algoritmo. La estrategia incremental se considera la más apropiada ya sea que la metodología de microingeniería se desarrolle como técnica de aprendizaje e inducción de los programadores noveles al arte-ciencia de la programación, o que sea aplicada por programadores experimentados en proyectos reales.

La etapa de Análisis tiene como objetivo identificar cada uno de los elementos de la tarea o problema a resolver. En un inicio se pueden clasificar en datos de salida o resultados, formulas y procesos, datos de entrada necesarios, condiciones aplicables al caso y las repeticiones requeridas en forma de ciclos.

La etapa de Diseño tiene por objetivo estructurar la secuencia de pasos para resolver el problema a través de diagramas de flujo, pseudocódigo y tablas de decisión combinando las diferentes estructuras de control que deben ordenarse, como piezas completas indivisibles y aplicando las reglas de la Lógica, para resolver el problema.

En la Implementación es conveniente hacer uso de algún lenguaje 3GL (incluso de varios simultáneamente) para poder expresar a través de la sintaxis específica de algún lenguaje la semántica de los diagramas de flujo o pseudocódigo utilizado.

La etapa de Pruebas debe incluir tanto pruebas de ejecución en computadora como pruebas de escritorio para identificar como la computadora ejecuta el procedimiento paso a paso en su memoria.

La Liberación se da cuando el problema o tarea tiene una solución computacional codificada, funcional y probada por el estudiante.

A continuación se hace una descripción detalla de cada una de las etapas y de los productos que se obtienen. Se presentará el modelo con un nivel completo de detalles. El programador experimentado puede obviar algunos de los pasos y avanzar rápidamente sobre las distintas etapas; pero el programador novato puede observar pormenorizadamente cada uno de los momentos de la construcción del algoritmo a nivel microscópico y construir su propia experiencia para poder, con la práctica explicada en la sección 3.4, desarrollar sus propias habilidades de forma progresiva.

3.4.2. Análisis

La etapa de análisis tiene como objetivo identificar cada uno de los elementos de la tarea o problema a resolver. El análisis, como puede observarse en la figura 40, identifica los componentes del problema en seis categorías: resultados, procesos, entradas, condiciones iteraciones y diccionario de datos. Éstos son los productos finales del análisis.

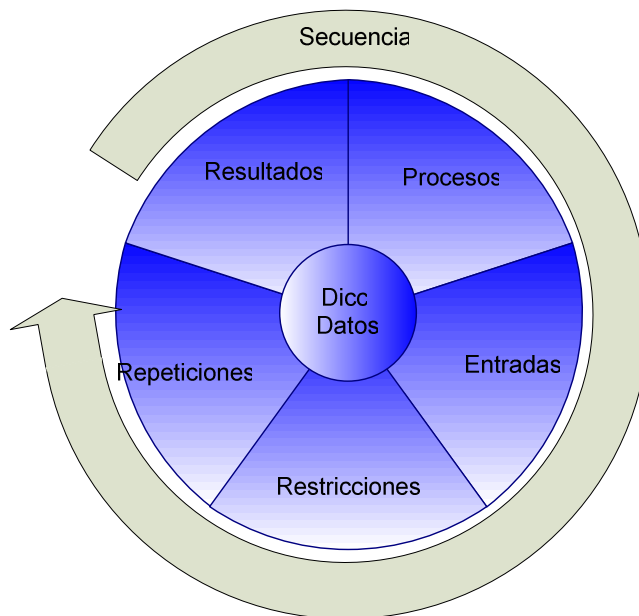


Figura 40. Modelo del análisis

La estrategia básica propuesta sobre la que opera este modelo de análisis es la de retroceso (backwards) la cual consiste en iniciar por el producto final para descubrir progresivamente cada uno de los demás elementos. Esté secuencia se explica a continuación:

1. Se inicia por la identificación de los resultados que se deben alcanzar, es decir, el producto final.

2. El siguiente paso es la identificación de los procesos que son necesarios para producir los resultados esperados.
3. Estos procesos arrojan luz para poder identificar los datos de entrada requeridos para hacer sus cálculos u operaciones.
4. Posteriormente se identifican las restricciones del mundo real y las del mundo virtual.
5. Cuando un proceso es necesario repetirlo se marca como repetición.
6. Todos los datos utilizados sean de entradas, de proceso, de salida o auxiliares se integran en el diccionario de datos.

Combinando la estrategia incremental (presentada en la espiral la metodología) a través de iteraciones sucesivas, con la de descubrimiento hacia atrás o retroceso (de la etapa de análisis) se logra un proceso de aproximación progresivo y reflexivo que es el objetivo que se busca desde un inicio. Más adelante en este capítulo se presenta un caso de estudio.

En cada iteración de la espiral se refinan y enriquecen los productos finales del análisis. Por ejemplo, en una primera iteración se pueden ejecutar los pasos 1, 2, 3 y 6 para obtener una primera versión simple (considerando únicamente salidas, procesos y entradas, excluyendo las restricciones y las repeticiones) que pasara a diseño, codificación, pruebas y liberación; en una segunda iteración se refinan los elementos ya identificados en la primera iteración y se agregan elementos del paso cuatro (en esta iteración se agregarían las restricciones) para tener un nuevo diseño, código y pruebas; en una tercera iteración se agregan los elementos del paso 5 (las repeticiones) y se completa el ciclo cumpliendo con las siguientes etapas; y así sucesivamente hasta contar con una versión completa.

Para facilitar el proceso de análisis y poder obtener los seis productos ya explicados, se considera el uso de las siguientes herramientas:

- **Árbol de descomposición de procesos.**- Es un dígrafo arborescente que muestra la secuencia de cálculo necesaria para obtener un resultado final. En la figura 41 puede verse un ejemplo en cual las entradas de datos básicos, llamados insumos, inician en las hojas del árbol y se combinan para crear productos parciales, parciales y llegar a la parte superior creando el producto final o resultado del proceso. Este árbol es de orden n y no está balanceado.
- **Tablas de decisiones.**- En el capítulo dos se explica esta herramienta a través de la figura 20.
- **Árboles de decisiones.**- En el capítulo dos se explica esta herramienta a través de la figura 21.

Al concluir cada iteración del análisis se obtienen los siguientes productos:

1. Listado de resultados esperados.
2. Listado de procesos acompañado de una descripción de los mismos. El árbol de descomposición de procesos es una herramienta útil en esta etapa.
3. Listado de datos de entrada. Las hojas de insumos del árbol de descomposición de procesos arrojan esta información.
4. Relación de restricciones del mundo real y del mundo virtual. Los árboles de decisiones y las tablas de decisiones son herramientas de aplicación natural en esta etapa.

5. Listado de repeticiones.
6. Diccionario de datos.

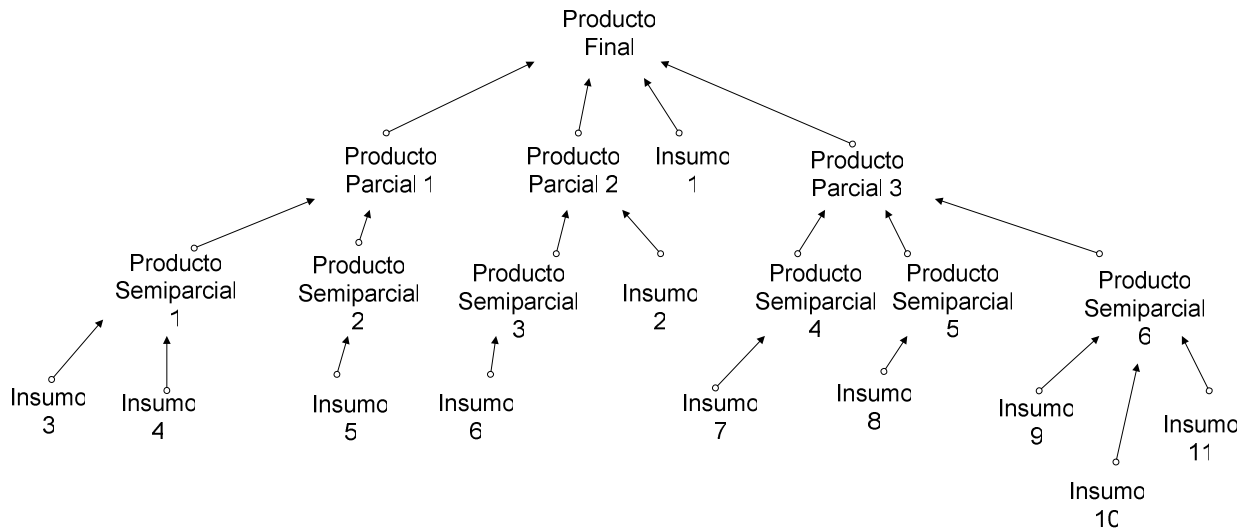


Figura 41. Árbol de descomposición de procesos

3.4.2.1. Resultados

Un algoritmo puede producir una vasta variedad de resultados, también llamadas salidas. Estas pueden ser uno o varios datos resultados de cálculos matemáticos; la creación de una tabla, archivo, reporte u otro conjunto de datos; el movimiento de un brazo mecánico robotizado; la reproducción de audio o video en un dispositivo de salida; etc.

El objetivo de esta etapa es crear un listado de todas las salidas explicándolas e identificando el lugar dónde se produce la salida. Este listado puede tener la forma de la tabla 6.

Salida	Descripción	Lugar
areaTotal	Área total del jardín a ser empastado resultado de restar el área del terreno menos el área de la fuente central	Monitor
beep	Al terminar el proceso de cálculo el sistema emite un beep para notificar al usuario.	Bocinas
reporteAlumnos	Reporte de estudiantes inscritos a la materia de principios de programación. Incluye la matrícula y el nombre	Impresora

Tabla 6. Tabla de resultados

3.4.2.2. Procesos

Para la identificación de los procesos se toma cada una de las salidas de la Tabla de Resultados (ver Tabla 6) y se construye su Árbol de Descomposición de Procesos explicado en la figura 31 de la página anterior.

Por cada árbol de descomposición de procesos se realiza el siguiente tratamiento:

1. Se toma el producto de más bajo nivel y más a la izquierda del árbol y se transcribe a notación algorítmica la combinación de los insumos para producir el producto requerido. Utilizando la figura 31 se tendría que:

$$\text{productoSemiparcial1} = \text{insumo3} \theta \text{ insumo4}$$

Dónde el operador θ representa cualquier función u operador aritmético.

2. Este proceso se repite con el producto que se sigue a su derecha. Siguiendo con el ejemplo se tendría:

$$\text{productoSemiparcial2} = \theta \text{ insumo5}$$

3. Estos resultados parciales ahora se consideran insumos del nivel superior de árbol y se repiten los pasos 1 y dos. Por ejemplo, para obtener el producto parcial 1 se tendría:

$$\text{productoParcial1} = \text{productoSemiparcial1} \theta \text{ productoSemiparcial2}$$

4. Se sube progresivamente en el árbol hasta llegar a la raíz terminando el proceso. En el ejemplo se obtendría el producto final de la siguiente forma:

$$\text{productoFinal} = \text{productoParcial1} \theta \text{ productoParcial2} \theta \text{ insumo1} \theta \text{ productoParcial3}$$

El hecho de iniciar por el extremo inferior izquierdo y avanzar a la derecha y hacia arriba permite transcribir el árbol de descomposición de procesos al orden real de ejecución algorítmica ya que los primeros cálculos escritos corresponderán a las hojas y el último al de la raíz que representa el resultado último del proceso.

3.4.2.3. Datos de Entrada

Los datos de entrada se identifican de forma clara en los árboles de descomposición de procesos ya que forman las hojas del árbol y se definen como insumos.

Los datos de entrada pueden provenir de distintas fuentes, podrían ser datos aportados por el usuario, datos generados aleatoriamente, datos leídos de una base de datos, datos obtenidos a través de sensores físicos de algún mecanismo, datos escaneados y observados a través de cámaras, etc.

El producto final de este proceso es identificar y crear un listado de todos los datos de entrada que son necesarios para poder obtener el resultado identificado en la primera etapa. Las entradas pueden concentrarse en una Tabla de Entradas como la mostrada en la Tabla 7.

Entrada	Descripción	Fuente
radioFuente	Radio de la fuente central del jardín que se empastará.	Usuario
temperatura	Temperatura en tiempo real de un horno	Sensor
respuesta	Alveolo relleno en un test por parte de un estudiante dando respuesta a un reactivo	Escáner

Tabla 7. Tabla de entradas

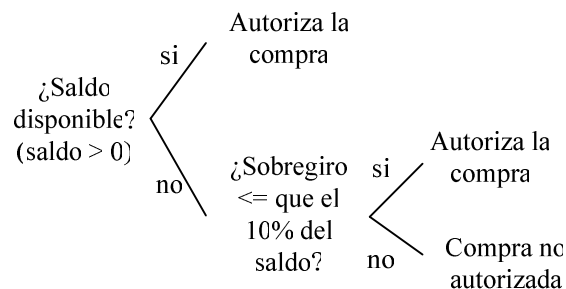
3.4.2.4. Restricciones

Las restricciones son condiciones que los datos deben de satisfacer para poder ser considerados como válidos para el proceso. Estas restricciones pueden pertenecer al dominio del problema (mundo real) o al dominio de la solución (mundo virtual). Las restricciones se escriben en forma de condición aplicando el dominio al que pertenece.

Para poder identificar las restricciones se realizan los siguientes pasos utilizando los datos definidos en el diccionario:

1. Se definen los dominios para cada una de las variables incluidas en el diccionario de datos.
2. Los dominios definidos para las variables de entrada se les considera condiciones de control candidatas a procesos de validación por medio de ciclos abiertos. Esto se hace en función del criterio de usabilidad. Cuando una variable de entrada es errónea debe enviarse un mensaje descriptivo al usuario y permitirle de nuevo la entrada del dato, es decir se repite la lectura tantas veces como sea necesario hasta tener los datos correctos.
3. Los dominios definidos para las variables de procesos se validan contra restricciones impuestas por el mundo real y se les considera candidatos a procesos de decisión por medio de condicionales. No todas las variables deben tener estructura condicional ya que sus escenarios de validez pueden haber sido creados como consecuencia de validar las entradas. En caso de si requerir la estructura condicional, ésta determina las acciones a seguir bajo cada posible evaluación de verdadero o falso de la condición. Para poder representarlas se pueden utilizar árboles de decisiones como se explica en la sección 2.2.4, en la figura 21; en forma de tablas de decisiones, explicadas en la misma sección, figura 20; o a través de los diagramas de Nassi/Shneiderman explicados también en la sección 2.2.4 en la figura 24.

A continuación se presenta un ejemplo:



Observe que la restricción se escribe a través de una condición cuya evaluación únicamente puede obtener resultados binarios del tipo si o no.

4. Los dominios definidos para variables de salida pueden estar garantizados como consecuencia de aplicar las restricciones de las variables de entrada y de proceso por lo

tanto no son candidatas inmediata ni a estructuras de ciclos ni de condiciones. Si el caso los amerita se define su estructura condicional y de repetición.

En conclusión: Las variables de entrada se definen sus condiciones para controlar estructuras de ciclos abiertos; las de proceso se pueden convertir en estructuras de tipo condicional; y las de salida en general se satisfacen sus dominios como consecuencia de satisfacer los dos primeros tipos de datos.

3.4.2.5. Repeticiones.

Se generan cuando se identifica que cierto número de procesos se repite varias veces. Se pueden representar a través de los diagramas de Nassi/Shneiderman explicados en la sección 2.2.4, figura 24 o a través de una sencilla narrativa.

Las repeticiones deben de terminar en algún momento para lo cual se debe definir el límite de iteraciones a realizar. Este límite puede describirse en forma de condición o en un número fijo de iteraciones.

Por ejemplo:

- Se repite la lectura del radio de la fuente mientras el radio sea menor a cero (cuando sea mayor a cero entonces es un dato correcto).
- Se repite el cálculo del área de la pared cuatro veces (porque el cuarto tiene cuatro paredes)

3.4.2.6. Diccionario de Datos

El diccionario de datos es un documento (ver tabla 8) que concentra todos los datos que el algoritmo requiera. Para su obtención se puede utilizar la información recopilada de la identificación de resultados, de procesos y de entradas.

Nombre	Descripción	Tipo	Categoría	Dominio
presupuesto	Presupuesto total de sembrar pasto en el jardín	Real	Salida	Presupuesto > 0
tmc	Total de metros cuadrados de pasto requeridos	Real	Salida	tmc > 0
mo	Mano de Obra de colocación por metro	Real	Entrada	mo > 0
areaRectangulo	Total de metros cuadrados del rectángulo en donde se sembrará el pasto	Real	Proceso	areaRectangulo > 0
areaCirculo	Total de metros cuadrados	Real	Proceso	0 < areaCirculo < areaRectangulo
radio	Radio de la fuente central	Real	Entrada	radio > 0
costoPasto	Costo del metro cuadrado de pasto	Real	Entrada	costoPasto > 0

Tabla 8. Diccionario de datos

La tabla estará formada por una columna con el nombre del dato, otra para su descripción, una más para identificar el tipo de dato, otra columna es para explicar la categoría a la que pertenece el dato y finalmente una última columna para definir su dominio. A continuación se presenta un ejemplo de Diccionario de Datos.

3.4.3. Diseño

Ian Sommerville en [98] dijo que la esencia del diseño es tomar decisiones sobre la organización lógica del software. Roger Pressman en [87] define al diseño como el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo.

Al concluir el análisis se cuenta con todas las partes que forman al problema a resolver clasificadas en resultados, procesos, entradas, restricciones y repeticiones; esto acompañado de un diccionario de datos que concentra las definiciones, tipos y dominios de cada uno. El diseño consiste en trazar una ruta, un especie de “camino” que conecta todos los elementos del análisis y que hay que transitar para poder llegar a la meta. En la figura 42 se muestra conceptualmente el diseño de esta ruta. Cuando el código esté terminado, la computadora recorrerá este camino partiendo del origen, decidiendo, según el valor de los datos que tiene, sobre rutas alternativas, repitiendo en forma de círculos procesos que deben hacerse cíclicamente para finalmente llegar a su destino.

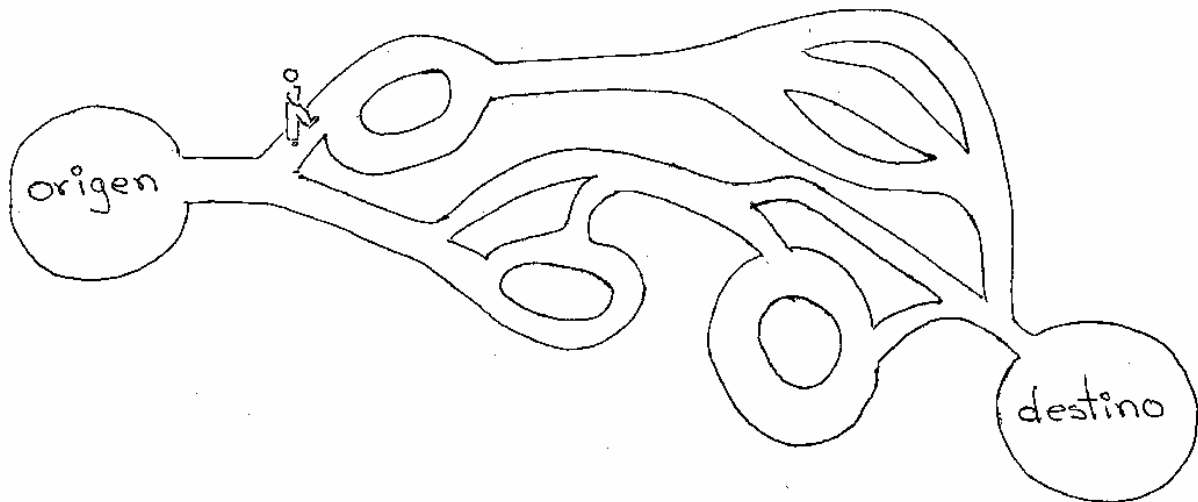


Figura 42. Diseño conceptual del viaje de la computadora

A continuación, en la figura 43 se muestran las cuatro etapas que se proponen en la metodología de microingeniería para la elaboración del diseño. En ella se identifica que se inicia el diseño por la base para después subir en la pirámide. Debido a que se considera que el proceso se repite en cada iteración de la espiral de la metodología se realiza la integración y refinamiento en cada etapa de diseño ascendente.

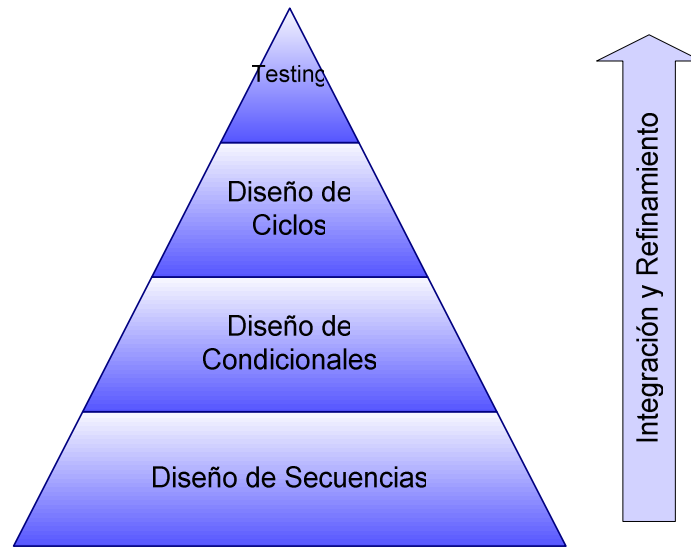


Figura 43. Modelo de diseño de algoritmos

Para poder completar con éxito el proceso de diseño se tiene como punto de partida los productos resultados de la etapa de análisis. Así cada producto parcial del diseño, resultado de sus cuatro etapas, es consecuencia natural de su correspondiente producto del análisis. En la figura 44 se muestra el tránsito que se tiene entre estos dos grandes momentos:

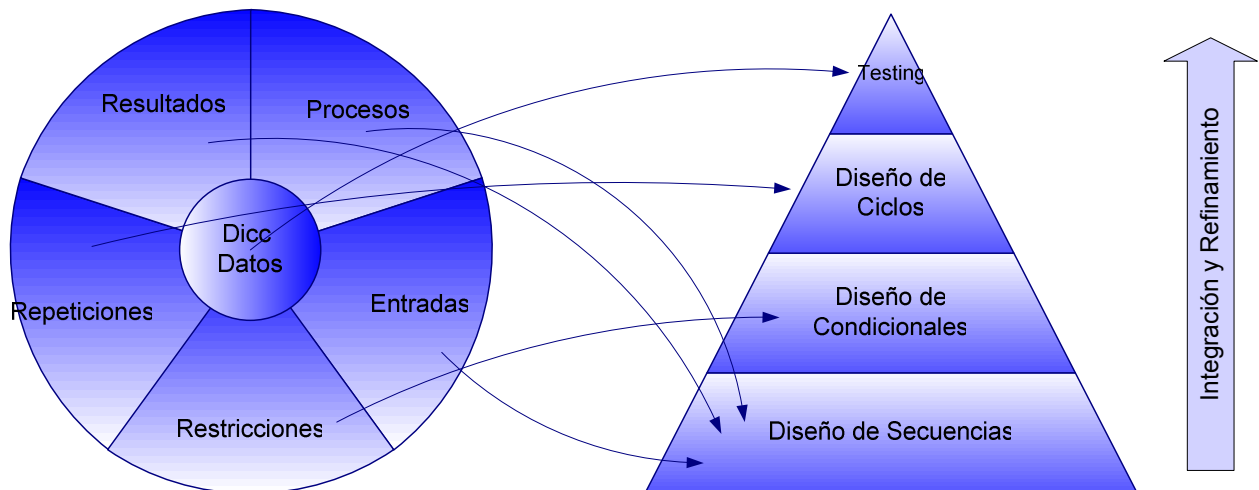


Figura 44. Modelo de tránsito del análisis al diseño

La metodología de microingeniería propone el siguiente modelo de tránsito del análisis al diseño. Los pasos son los siguientes:

1. Tomar cada uno de los resultados deseados y sus procesos asociados y preparar los diagramas de secuencia (entrada-proceso-salida) sin ningún otro componente. Pueden ser varios diagramas de secuencia separados.
2. Realizar una primera integración aplicando las reglas de orden, existencia, etc. (heurísticas explicadas en la sección 3.3).
3. Revisar que el modelo integrado cumpla con el primer resultado esperado.
4. Agregar las condiciones, restricciones y validaciones a los diagramas individuales y luego integrarlos de nuevo en uno solo. Se pueden utilizar las estrategias de Soloway [96]
 - a. Concatenar Plan
 - b. Anidar Plan
 - c. Fusionar Plan
 - d. Adaptar Plan.
5. Revisar que el segundo modelo cumpla con los resultados
6. Agregar las repeticiones o iteraciones identificadas a cada diagrama de secuencias + condiciones.
7. Realizar una nueva integración y revisar los resultados
8. Continuar con un proceso de pruebas y refinamiento hasta completar todas las especificaciones obtenidas del análisis.

La metodología debe permitirnos modelar la solución de un problema algorítmicamente utilizando algún lenguaje de modelado (por ejemplo diagramas de flujo o el lenguaje propuesto por esta tesis más adelante en la sección 3.5.1) aplicando las heurísticas como guías de diseño.

No se debe perder de vista que en este momento se está aplicando Lógica de Programación y no análisis y diseño de sistemas ni tampoco ingeniería de software, sin embargo el incluir desde esta temprana etapa de la formación de un futuro profesionalista las etapas de análisis, diseño, construcción, pruebas y liberación permitirá eventualmente enfrentar los temas de ingeniería de software como una secuencia natural de la espiral de aprendizaje.

Por último, no se debe olvidar que la Lógica de Programación es una habilidad mental que debe desarrollarse a través de experiencias; pero que estas deben ser planeadas en un orden de complejidad y de temática y que deben estar vinculadas tanto con su realidad, como con otras materias.

A continuación se describen con más detalle cada una de las etapas de diseño. En la sección 3.4.7 se presenta un caso de estudio completo donde se puede apreciar la metodología de Microingeniería de software, el uso de un lenguaje visual propuesto y la aplicación de las heurísticas de diseño.

3.4.3.1. Diseño de Secuencias

Una secuencia es una estructura de control que contiene un conjunto de instrucciones que se ejecutan una inmediatamente después de la otra y típicamente incluyen la entrada, el proceso y al

salida de ciertos resultados. Cada resultado tiene su propia secuencia de cálculo por lo que se podrían crear varios diseños de secuencias, al menos uno precisamente para cada resultado.

Para la realización del diseño de secuencias se toma como punto de partida los productos creados en las etapas de resultados, procesos y entradas del análisis. El árbol de descomposición de procesos contiene los elementos y sus relaciones así que se toma como fuente primaria de información para elaborar el diseño de secuencias.

En la figura 45 se observa un par de ejemplos de diseño de secuencias utilizando diagramas de flujo convencionales. Durante el proceso de diseño se aplican las heurísticas que se explican en la sección 3.3. Éstas tienen por objetivo orientar sobre diseños correctos.

3.4.3.2. Diseño de Condicionales

Las condicionales son rutas alternativas que pueden recorrerse y la decisión del camino a tomar depende de evaluar una condición. Pueden ser de dos tipos:

- Condicional con un camino (tipo if-then)
- Condicional con dos caminos (tipo if-then-else)

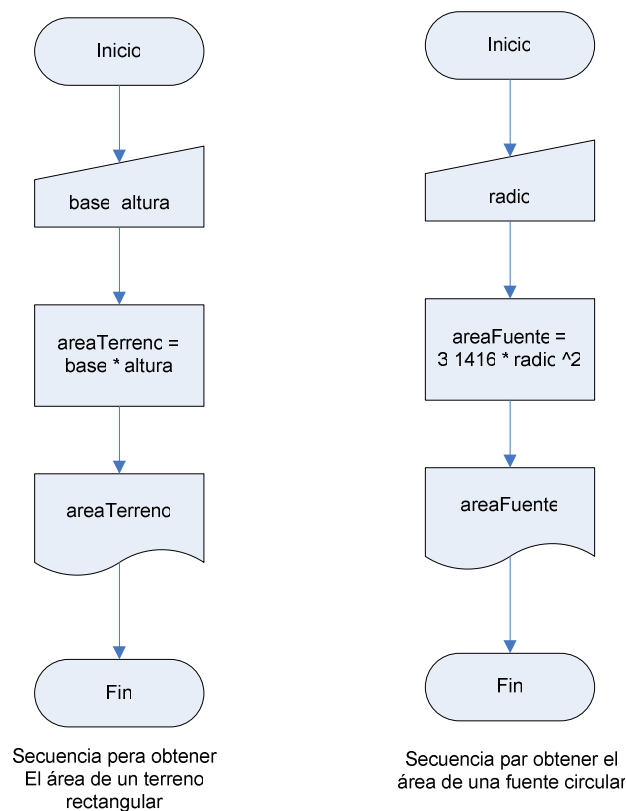


Figura 45. Ejemplos de diagramas de secuencias

La elección en el diseño de cual es la conveniente depende del número de rutas alternativas que se pueden crear al evaluar la condición. Cada ruta alternativa puede estar formada por secuencias, repeticiones o nuevas condicionales (ver lenguaje definido en la sección 3.4.1)

Una condicional y una condición no es lo mismo. La condicional es una estructura completa que incluye una condición y sus rutas alternativas de ejecución. Una condición es una expresión que al ser evaluada siempre produce un resultado booleano de tipo verdadero o falso.

Hay dos tipos de condiciones que se pueden utilizar en las condicionales:

- Condición simple es la comparación de dos datos a través de un operador relacional (<, >, =, etc.)
- Condición compuesta es la comparación ya sea entre dos condiciones simples, o entre dos valores lógicos, o la combinación de uno lógico con una condición simple a través de un operador lógico (AND, OR, XOR, etc.).

En la tabla 9 se muestran ejemplos de estas definiciones.

Condición Simple	Condición Compuesta
$a > 3$	$(\text{areaTerreno} \leq \text{areaFuente}) \text{ AND } (\text{precioPasto} \geq \text{presupuesto})$
$\text{areaTerreno} \leq \text{areaFuente}$	$\text{atorizacion OR } (\text{sobreGiro} < \text{margen})$

Tabla 9. Condición simple y condición compuesta

Existe un tercer tipo de condicional:

- Condicional con múltiples caminos (tipo case o switch)

Las condicionales con múltiples caminos no se construyen a partir de una condición sino de una variable ordinal y la ruta a seguir depende del valor que tenga esa variable.

La figura 46 muestra los posibles diseños de las condicionales

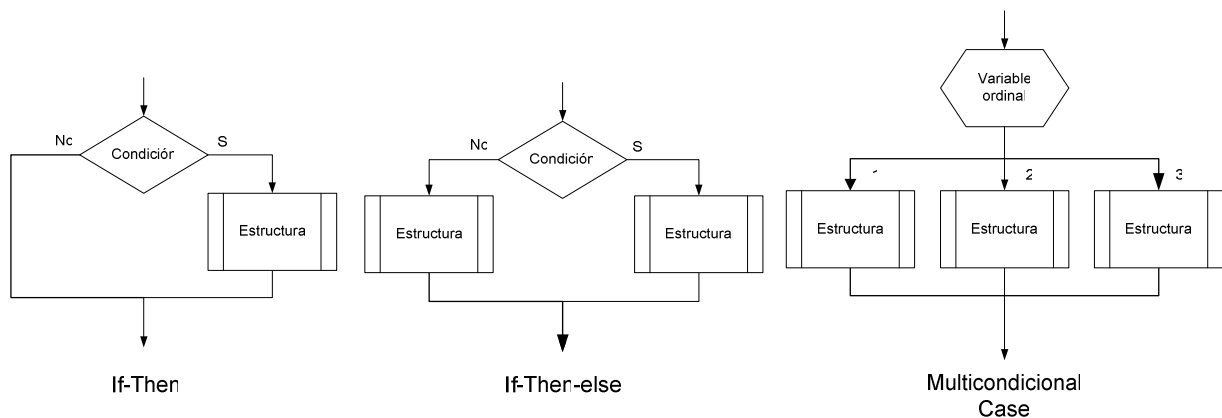


Figura 46. Diseños posibles de condicionales

3.4.3.3. Diseño de Ciclos

Un ciclo es una estructura que permite repetir la ejecución de otra estructura un número finito de veces. Hay dos categorías:

- Ciclo cerrado (for).- Se conoce de antemano el número de veces que se repetirá la ejecución de una estructura.
- Ciclo abierto.- Se desconoce el número de veces que se repetirá la ejecución de una estructura. Para poder determinar cuando terminar se utiliza una condición de control. La condición sigue las mismas reglas explicadas en la sección 3.4.3.2.

A su vez, los ciclos abiertos tienen dos versiones:

- Ciclo mientras (while).- Realiza la repetición mientras una condición sea verdadera. La condición se valida antes de decidir si ejecuta la estructura o no. Puede suceder que la estructura a repetir no se ejecute ni siquiera una vez.
- Ciclo hasta (until).- Realiza la repetición hasta que una condición se cumpla. La condición se valida después de haber ejecutado la estructura por lo que ésta se ejecutará siempre al menos en una ocasión.

Los ciclos abiertos requieren de una variable de control que se utiliza dentro de la condición. Esta variable debe cumplir con las dos siguientes reglas:

1. Debe tener valor válido antes de iniciar el ciclo. No hacerlo implica un proceso errático en la ejecución del ciclo.
2. Debe de cambiar de valor durante la ejecución del ciclo. No hacerlo implica el riesgo de un ciclo que nunca termine.

Para la elaboración del diseño de los ciclos se toma la información que hay en el análisis sobre la identificación de repeticiones. Dependiendo de si se sabe o no el número de veces de la repetición se decide si se aplica abierto o cerrado; Dependiendo de si la condición se escribe antes o después y de si la repetición debe ejecutarse al menos en una ocasión se decide entre ciclo mientras o ciclo hasta. Finalmente se dibuja la estructura del ciclo. En la figura 47 se puede observar ejemplos de los tipos de ciclos. La imagen representa si el ciclo es cerrado, abierto, con condición al inicio o con condición al final.

3.4.3.4. Diseño de Pruebas

El objetivo de realizar pruebas en un programa es el de detectar todo posible malfuncionamiento antes de considerar un algoritmo terminado. Esto aportará validez y confianza al algoritmo que se está diseñando. Probar un programa es ejercitarlo con la peor intención a fin de encontrarle fallos.

La cantidad de combinaciones de entradas posibles y su consecuente variación de ejecución representan un universo casi infinito de posibilidades para la ejecución de un algoritmo lo cual sería imposible de validar al 100%. Es por ello que es importante diseñar una estrategia clara y

práctica que se materialice en forma de un plan específico de pruebas. Este plan estará integrado por un conjunto de casos de prueba que deben ser diseñados en esta etapa.

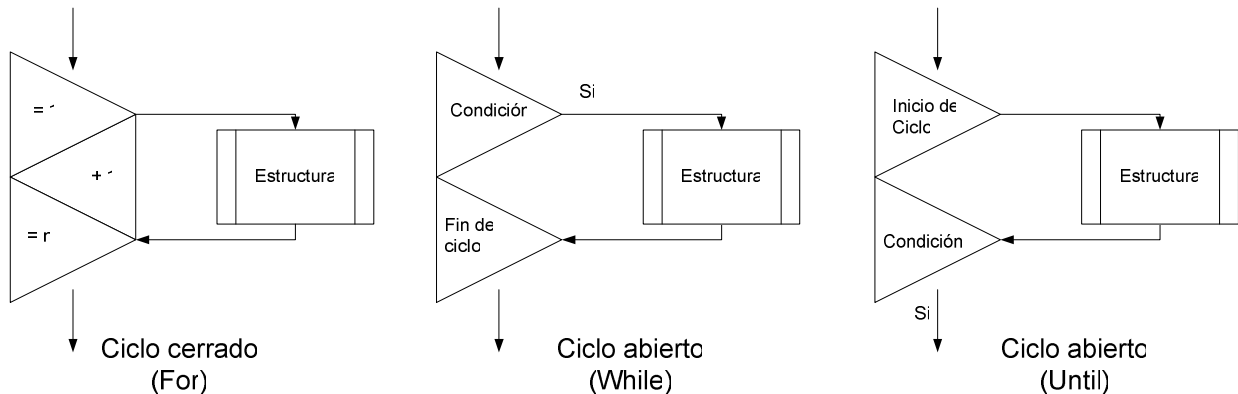


Figura 47. Diseños de ciclos

El diseño de un caso de prueba para este modelo consta de tres bloques de información:

- El propósito de la prueba
- El conjunto de datos de entrada (un valor por cada entrada identificada en la etapa de análisis)
- El resultado o resultados que se esperan (un valor por cada salida identificada en el análisis)

Existen muchas y variadas modalidades de pruebas, la metodología de Microingeniería considera al menos el diseño de dos tipos:

1. Pruebas de caja negra.- Se dice que una prueba es de caja negra cuando prescinde de los detalles del código y se limita a lo que se ve desde el exterior. Intenta descubrir casos y circunstancias en los que el algoritmo no hace lo que se espera de él.
2. Pruebas de caja blanca.- En oposición al término "caja negra" se suele denominar "caja blanca" al caso contrario, es decir, cuando lo que se mira con lupa es el código que está ahí escrito y se intenta que falle. También se le llama prueba estructural.

Como se mostró en la figura 44 (modelo de tránsito del análisis al diseño), las pruebas se diseñan en función del diccionario de datos que se definió en la etapa de análisis. Cada dato identificado está señalado si es de entrada, de proceso o de salida; al mismo tiempo se ha descrito su tipo y su dominio. Esto es particularmente útil para las pruebas de caja negra.

Para el proceso de diseño de pruebas de caja negra es necesario crear un conjunto de casos para cada resultado esperado. Cada caso, a su vez, estará formado por el conjunto de valores que se asignarán a los datos de entrada y el conjunto de valores de salidas que se esperan para esas entradas. Para la selección de cada dato del conjunto de datos de entrada de un caso se sugiere aplicar las siguientes reglas de diseño de caja negra:

- Si un parámetro de entrada debe estar comprendido en un cierto rango, aparecen tres clases de valores para pruebas: una por debajo del rango; una dentro del rango; y una por encima del rango.
- Si una entrada requiere un valor específico, también se tienen tres clases de valores para pruebas iguales al caso anterior.
- Si una entrada requiere un valor de entre los de un conjunto, se tienen dos clases de valores para pruebas: una con el valor perteneciente al conjunto; otra con un valor que no pertenezca al conjunto.
- Si una entrada es lógica (booleana), Se tienen únicamente dos posibilidades de valores para pruebas: verdadero y falso.
- Estos mismos criterios se aplican a las salidas esperadas: hay que intentar generar resultados en todas y cada una de las clases.

A continuación se presenta un par de ejemplos de diseño de pruebas de caja negra:

CASO 1 Caja Negra	Validar que el cálculo del área de un jardín que se va empastar sea correcto	
ENTRADAS: areaTerreno = 40 areaFuenteCentral = 3.7	SALIDAS: areaEmpastada = 36.3	

CASO 2 Caja Negra	Validar que el área de la fuente no sea mayor que la del jardín	
ENTRADAS: areaTerreno = 30 areaFuenteCentral = 37	SALIDAS: Error, No puede ser más grande la fuente que el terreno del jardín.	

En el caso de las pruebas de caja blanca, como ya se explicó, se aspira a encontrar errores del código. También se diseñarán los casos utilizando las recomendaciones ya expuestas pero además se considerarán casos específicos para probar cada una de las tres diferentes categorías de estructuras de control: secuencias, condiciones y repeticiones.

Validar las secuencias.- Como ya se explicó en la sección de Diseño de Secuencias, estas son un conjunto de instrucciones que típicamente llevan el orden de pedir datos, hacer cálculos y emitir salidas. Un algoritmo está integrado por una o más secuencias. Se puede preparar casos de pruebas para cada una de ellas utilizando las recomendaciones de valores de la sección anterior.

Validar las condiciones.- Las condiciones son el punto de partida para estas estructuras, al evaluarlas puede haber una alternativa de ejecución (if-then); dos alternativas de ejecución (if-then-else) o múltiples alternativas de ejecución (case). Un diseño de pruebas para este tipo de estructura debe considerar cada una de las múltiples rutas de ejecución y cada uno de los valores que forman la condición.

Validar las repeticiones.- Una repetición (o ciclo) ejecuta un conjunto de instrucciones un cierto número de veces; ese número de veces debe ser preciso y controlado. Como se explicó

antes, hay repeticiones abiertas de tipo “mientras” y “hasta” y cerradas de tipo “for”. Cada una de ellas debe ser probada bajo las siguientes

Para un ciclo de tipo mientras (while) hay que diseñar tres casos de pruebas para la condición de control:

- Ninguna ejecución de las instrucciones del ciclo
- Una ejecución de las instrucciones del ciclo
- Más de una ejecución de las condiciones del ciclo

Para un ciclo de tipo hasta (until) hay que diseñar dos casos pruebas para la condición de control:

- Una ejecución de las instrucciones del ciclo
- Más de una ejecución de las condiciones del ciclo

Para un ciclo de tipo for basta con diseñar un caso de prueba de una ejecución de las instrucciones del ciclo.

En los tres tipos de ciclos se recomienda, además, diseñar casos de prueba para los valores límite de la condición de control de la siguiente forma:

- Con un valor antes de cumplir la condición.
- Con el valor para satisfacer la condición de control.
- Con un valor después de cumplir la condición de control.

3.4.3.5. Integración y Refinamiento

En cada iteración en la espiral de la metodología se aplica un proceso de integración y refinamiento. Este consiste en conjuntar cada uno de los distintos diagramas de secuencia, condicionales y ciclos en uno nuevo diagrama que será revisado, depurado y probado.

Típicamente en la primera iteración de la metodología se omiten condicionales y ciclos así que únicamente se contará con varios diseños de secuencias que deben ser integrados, refinados y probados. En la segunda iteración ya se contará con los diagramas condicionales; en otra iteración más se tendrán las repeticiones. Este proceso iterativo de la espiral no se limita simplemente a tres iteraciones, se realizaran todas las necesarias hasta estar satisfechos con la versión final del algoritmo y programa. Cada iteración integrará los nuevos diseños al anterior hasta terminar.

Como ya se explicó, Soloway menciona que hay cuatro formas de combinar lo que él llamó planes:

- Concatenación.- Dos planes se juntan en secuencia, uno después del otro.
- Anidación.- Poner un plan dentro de otro.
- Fusión.- Combinar dos planes
- Adaptación.- Modificar un plan para adaptarlo a una situación específica.

Estas cuatro técnicas de integración son retomadas por la metodología de microingeniería para unir los distintos diagramas de secuencia, condicionales y ciclos. Cada uno de ellos está

formando una estructura y, de acuerdo a las heurísticas y al lenguaje propuesto, éstas deben ser integradas y depuradas. En la sección 3.4.7 se muestran ejemplos de las técnicas de integración como parte de la solución a un caso de ejemplo.

El refinamiento se realiza como parte del proceso de depuración y validación resultado de la aplicación de las pruebas. Tiene por objetivo obtener una mejor versión que la anterior, dónde mejor significa incrementos en las evaluaciones de usabilidad, mantenibilidad, confiabilidad, funcionalidad y eficiencia.⁶

3.4.4. Construcción

La etapa de construcción de la metodología de microingeniería no presenta diferencias con respecto a los métodos tradicionales. Todo cuanto hay que hacer es tomar el resultado del diseño y transcribirlo a un lenguaje de programación imperativo tipo 3GL.

Este proceso de transcripción del diseño a código se realiza a través de la traducción directa de las estructuras de control del lenguaje de modelado a instrucciones 3GL. En la sección 3.5 se explica una arquitectura propuesta para poder crear código imperativo 3GL en forma automatizada a través de mecanismos de ingeniería hacia delante de código completo (Full Code Forward Engineering).

La técnica de traducción se hace construyendo esqueletos completos y posteriormente rellenando los espacios de los bloques. Por ejemplo, en la figura 48 se muestra una estructura condicional anidada en formato de diagrama de flujo. El procedimiento recomendado es:

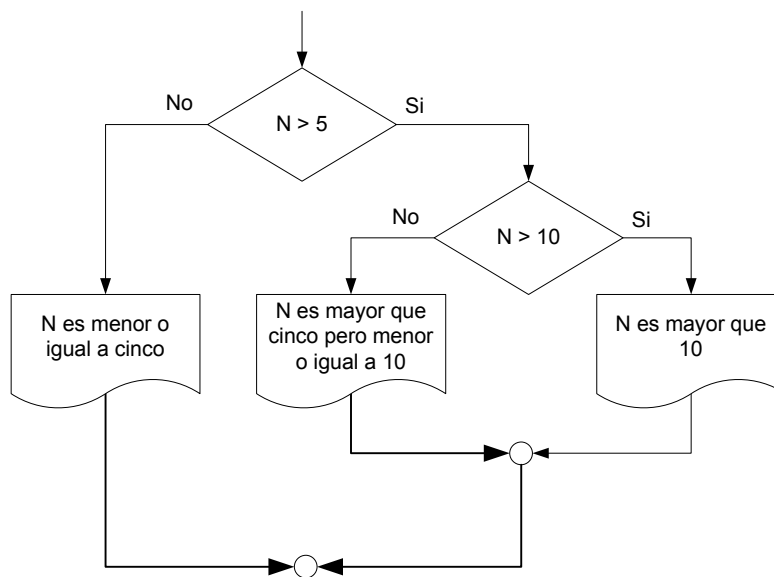


Figura 48. Diseño de bifurcaciones anidadas

⁶ Estos criterios corresponden a la norma ISO 9126 de calidad de software eliminando el de portabilidad por no ser aplicable al caso de estudio.

1. Escribir el esqueleto de primer condicional respetando las recomendaciones del uso de sangrías.
2. Escribir el esqueleto del esqueleto del condicional anidado en su lugar correcto
3. Completar el código perteneciente a cada bloque de código.

1er paso	2° paso	3er paso
<pre>if (n > 5) then begin end else begin end end</pre>	<pre>if (n > 5) then begin if (n > 10) then begin end else begin end end end else begin end end</pre>	<pre>if (n > 5) then begin if (n > 10) then begin Write("n es mayor que 10"); end else begin Write("n es mayor que cinco pero menor o igual a 10"); end end end else begin Write("n es menor o igual a cinco"); end end</pre>

Tabla 10. Ejemplo de transcripción a Pascal

1er paso	2° paso	3er paso
<pre>if (n > 5) { } else { }</pre>	<pre>if (n > 5) { if (n > 5) { } else { } } else { }</pre>	<pre>if (n > 5) { if (n > 10) { Console. Write("n es mayor que 10"); } else { Console. Write("n es mayor que cinco pero menor o igual a 10"); } } else { Console. Write("n es menor o igual a cinco"); }</pre>

Tabla 11. Ejemplo de transcripción a C#

Esta técnica permite tener control sobre la apertura y cierre de los bloques que forman cada estructura y evitan la confusión sobre la correspondencia de cada fin de bloque con su estructura. Siguiendo este proceso concluirá la etapa de construcción con el listado del algoritmo en código listo para ser compilado y ejecutado por la computadora.

Por supuesto, el uso de los depuradores y otras ayudas del IDE del lenguaje son ampliamente recomendados.

Para los nombre de identificadores se recomienda aplicar la nomenclatura húngara para definir los nombres de las clases, métodos, propiedades o variables necesarias del algoritmo. Esta recomendación debe ser aplicada desde que se inicia la iteración de la metodología en la etapa de análisis.

Un software desarrollado bajo la arquitectura de Diseño de Algoritmos Asistido por Computadora (DAAC), uno de los componentes de esta tesis, permitirá la creación de código, en primera instancia 3GL para poder entrar a un compilador o en segunda instancia crear el código ejecutable directamente del diseño sin pasar por el lenguaje imperativo.

3.4.5. Aplicación de Pruebas

En esta etapa se ejecutan las pruebas diseñadas previamente. Su aplicación se hace ingresando el conjunto de datos de entrada de cada caso de prueba diseñado en la etapa de diseño y comparando los resultados obtenidos con los esperados.

Primero se realizarán las pruebas de caja negra. Estas ayudarán a determinar la eficacia del algoritmo. Posteriormente se aplicarán las pruebas de caja blanca. Éstas ayudarán a determinar la eficiencia del mismo.

La forma de aplicar una prueba de caja negra es ejecutando el programa y aportando los valores de cada conjunto de datos de cada caso. Se probarán todos los casos uno por uno comparando el resultado del programa con los resultados esperados planeados en el diseño de la prueba. En caso de no obtener resultados correctos se revisa tanto la transcripción del diseño al código como el diseño mismo. Las pruebas de caja blanca aportan luz para detectar este tipo de error.

Las pruebas de caja blanca pueden ser realizadas apoyándose en los depuradores de paso a paso o con lápiz y papel utilizando la técnica de la corrida de mesa. Se aplicarán todos los casos de prueba planeados en la etapa de diseño. En caso de no obtener los resultados esperados también de revisa la transcripción del diseño a código y se valida el diseño con las observaciones realizadas a durante la aplicación de esta prueba.

Cuando se realiza un algoritmo se espera que este funcione, sin embargo, como resultado de la aplicación de pruebas se detectarán los errores y vicios que puede tener el algoritmo. Una vez identificados los errores se procede a su corrección.

Para poder corregir un error lo primero que se debe hacer es identificarlo. Para ello las pruebas de caja negra aportan un primer indicio del problema y las pruebas de caja blanca permiten la identificación clara del origen de este. Una vez identificado el tipo de error se procede a su corrección. En [27] se presenta un estudio Argentino sobre la clasificación de los errores y estrategias de solución.

3.4.6. Liberación

La etapa de liberación simplemente consiste en la entrega del programa probado y funcionando como resultado de haber realizado exitosamente cada una de las etapas anteriores así como de toda la documentación creada a lo largo del ciclo.

Ya que la metodología de microingeniería es en espiral se realizarán varias liberaciones, una por cada iteración de la espiral. Cada liberación corresponde a una versión distinta que deberá ser identificada claramente para evitar futuras confusiones. Cada liberación incluye la documentación completa de su ciclo incluyendo los productos de su análisis; los diseños realizados con su correspondiente integración y refinamiento; la codificación en su lenguaje 3GL; y finalmente la aplicación y resultado de sus pruebas así como de las acciones tomadas para corregir los errores detectados.

En forma típica, la primera liberación (o versión) corresponde a un programa que realiza unos pocos cálculos de la funcionalidad completa que se requiere y sin ningún tipo de validación o control. Conforme las iteraciones se realicen habrá nuevas versiones cada vez más completas y robustas.

El control de las versiones puede realizarse a través de un número entero acompañado de uno o varios decimales. La porción entera puede representar el número de iteración y la parte decimal el consecutivo de versiones que se desarrollaron hasta satisfacer las pruebas. Sin embargo, esto es únicamente una sugerencia. Lo importante es poder controlar la versión con su documentación para poder diferenciarla de las demás.

3.4.7. Caso de Estudio

En esta sección se presenta un ejemplo a partir del cual se aplica la metodología de microingeniería, las heurísticas y un lenguaje de modelado explicado en la sección 3.5.1. Por supuesto se considera necesario haber ejecutado la metodología de desarrollo de habilidades, explicada en la sección 3.6 para poder elaborar cada acción con pericia.

3.4.7.1. Presentación del Problema: El Jardín

La presentación del problema a resolver se redacta siguiendo las reglas del Aprendizaje Basado en Problemas (PBL) en su modalidad de escenario:

Pedro es jardinero y le han pedido hacer un presupuesto para poner pasto en una casa que se está construyendo. El jardín es rectangular pero tiene una gran fuente circular en el centro. Pedro tiene que decirle a su cliente cuantos metros cuadrados de pasto tendrá que colocar y el importe total. Además Pedro cobra una cantidad por colocación que corresponde a su mano de obra. Alma es sobrina de Pedro y le dijo que podía hacerle un programa para que le ayudara a calcular lo que le piden.

3.4.7.2. 1ª Iteración

En la primera iteración, como en todas, se ejecutan el análisis, diseño, construcción, pruebas y liberación, pero en cada fase únicamente se realizan algunos pasos que poco a poco se van incorporando al producto final

3.4.7.2.1. Fase de Análisis

De acuerdo a la metodología se inicia en el análisis con una estrategia backwards identificando primero las salidas, luego los procesos para continuar con las entradas, las restricciones y las repeticiones. En la primera iteración de la metodología se sugiere que únicamente se identifiquen en este análisis los tres primeros componentes: salidas, procesos y entradas, dejando para otra iteración las restricciones.

3.4.7.2.1.1. Identificación de Salidas

Al realizar una lectura detenida del problema se encuentra que se requiere como salida dos cosas:

- Presupuesto para poner pasto en un jardín
- Total de metros cuadrados a Sembrar

El formato del producto final se muestra en la tabla 12.

Salida	Descripción	Lugar
presupuesto	Presupuesto total de sembrar pasto en el jardín	Monitor
tmc	Total de metros cuadrados de pasto requeridos	Monitor

Tabla 12. Tabla de salidas.

3.4.7.2.1.2. Identificación de Procesos

La identificación de procesos consiste en observar que cálculos o procedimientos deben realizarse para poder generar las salidas propuestas en la etapa anterior. Dependiendo del nivel de habilidad que se haya desarrollado puede escribirse directamente la formulación algorítmica pero si aún se está iniciando se puede utilizar el árbol de descomposición como ayuda para poder definir la secuencia.

Según la metodología, la elaboración de un árbol es para cada salida identificada. En la figura 49 se observa el árbol de descomposición de procesos que se hizo para el presupuesto en la cual se define que el presupuesto se obtiene sumando el costo total de mano de obra y el costo total del pasto. A su vez, el costo total de mano de obra es el resultado de multiplicar el total de metros cuadrados de pasto por el precio por mano de obra que cobra el jardinero. Finalmente, el costo total de pasto se obtiene multiplicando el total de metros cuadrados de pasto por el costo del pasto. No se considera requerida la elaboración del árbol pero puede ser útil para visualizar el

mecanismo de obtención del cálculo. Si se ha desarrollado la habilidad (como se explica en la sección 3.6) puede obviarse y redactar directamente las formulas o procesos requeridos.

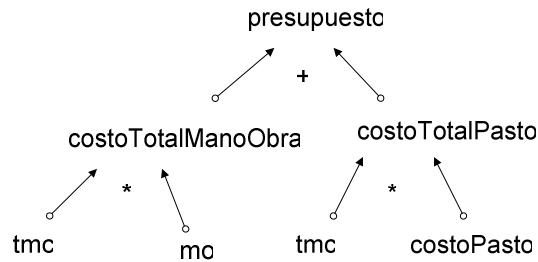


Figura 49. Árbol de descomposición de procesos para presupuesto

Siguiendo el mismo ejemplo, en la figura 50 se observa el árbol de descomposición de procesos para obtener el total de metros cuadrados de pasto. Este se calcula restandole al área del terreno completo (el rectángulo) el área de la fuente central (el círculo). A su vez, el rectángulo se obtiene multiplicando la base por la altura y el círculo Pi por el cuadrado del radio.

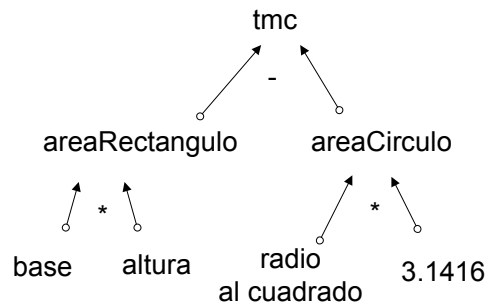


Figura 50. Árbol de descomposición de procesos para tmc

Con este análisis se pueden escribir las siguientes formulas como producto final de la parte de procesos:

$$\text{presupuesto} = \text{costoTotalManoObra} + \text{costoTotalPasto}$$

$$\text{costoTotalManoObra} = \text{tmc} * \text{mo}$$

$$\text{costoTotalPasto} = \text{tmc} * \text{costoPasto}$$

$$\text{tmc} = \text{areaRectangulo} - \text{areaCirculo}$$

$$\text{areaRectangulo} = \text{base} * \text{altura}$$

$$\text{areaCirculo} = 3.1416 * \text{radio} * \text{radio}$$

3.4.7.2.1.3. Identificación de Entradas

Siguiendo la metodología propuesta se identifican las entradas a través de localizar las hojas terminales (insumos) de los árboles creados, si los árboles no se elaboraron y se redactaron directamente las formulas se pueden obtener también de ahí observando los datos que requiere cada formula.

Del primer árbol se observa que la mano de obra y el costo del pasto son terminales, por lo tanto serán datos de entrada. El total de metros cuadrados, por su parte, no es terminal pues tiene su propio árbol de descomposición del cual, a su vez, se obtienen nuevos datos terminales que son la base, la altura y el radio. El valor de Pi no es un valor de entrada pues es una constante con valor conocido y no hay necesidad de leerla de ninguna parte.

El formato de entradas se muestra en la tabla 13.

Entrada	Descripción	Fuente
mo	Costo de la Mano de Obra por metro cuadrado de colocación de pasto.	Usuario
costoPasto	Costo del metro cuadrado de pasto	Usuario
base	Lado más largo del terreno donde se sembrará el pasto (base)	Usuario
altura	Lado más corto del terreno donde se sembrará el pasto (lado)	Usuario
radio	Radio del círculo que forma la fuente que se colocará al centro del terreno	Usuario

Tabla 13. Tabla en entradas

3.4.7.2.1.4. Diccionario de Datos

La elaboración del diccionario de datos se va realizando progresivamente durante las etapas de la fase de análisis para utilizarse en la fase de diseño y de construcción definiendo todas las variables a utilizar oportunamente.

Esta primera versión de diccionario de datos (tabla 14) omite la columna de dominio de los datos puesto que aún no se han identificado las restricciones.

Nombre	Descripción	Tipo	Categoría
presupuesto	Presupuesto total de sembrar pasto en el jardín	Real	Salida
tmc	Total de metros cuadrados de pasto requeridos	Real	Salida
mo	Mano de Obra de colocación por metro	Real	Entrada
areaRectangulo	Total de metros cuadrados del rectángulo en donde se sembrará el pasto	Real	Proceso
areaCirculo	Total de metros cuadrados	Real	Proceso
costoTotalManoObra	Costo total de la mano de obra necesaria	Real	Proceso
costoTotalPasto	Costo tota de los metros cuadrados de pasto a comprar	Real	Proceso
base	Lado más largo del rectángulo del terreno dónde se sembrará el jardín	Real	Entrada
altura	Lado más corto del terreno dónde se sembrará el jardín	Real	Entrada
radio	Radio de la fuente central	Real	Entrada
costoPasto	Costo del metro cuadrado de pasto	Real	Entrada

Tabla 14. Diccionario de datos 1a 1ª versión

Con esto concluye la etapa de análisis de la 1ª iteración y se inicia la etapa de diseño

3.4.7.2.2. Fase de Diseño

En la fase de Diseño se elabora el modelado del algoritmo que resuelve el problema a la luz de los productos obtenidos en el análisis. Siguiendo la metodología de microingeniería, en esta primera iteración solo se realizarán el diseño de secuencias y el diseño de casos de prueba en ese orden.

3.4.7.2.2.1. Diseño de Secuencias

Acorde con la metodología, el primer paso será el diseño de los diagramas de secuencia para cada una de las salidas requeridas. Se utilizará la versión modificada de diagramas de flujo explicada en la sección 3.5.1 como lenguaje de modelado.

En la figura 51 se observa la aplicación de la metodología de Microingeniería y la aplicación de algunas de las heurísticas llamadas Reglas de la Lógica.

1er paso.- En esta etapa se toma como modelo inicial un diagrama básico de tarea en el que se señala su inicio, su fin y una estructura de control por definir.

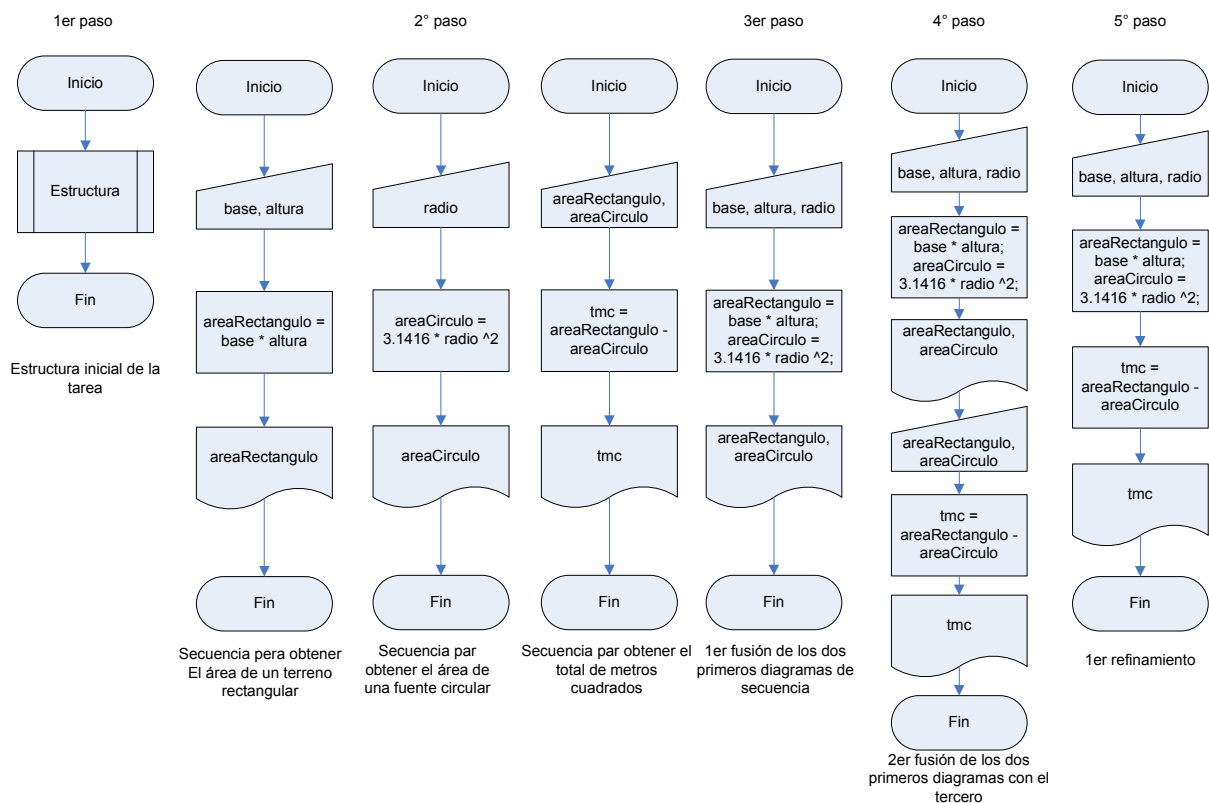


Figura 51. Primeros diagramas de secuencias

2º paso.- Utilizando las reglas cinco y seis (cerradura y creación), el diagrama de tarea inicial es sustituido por uno nuevo que, además, aplica la regla uno (secuencia) para proponer dos

entradas, un proceso y una salida. Este primer diagrama de secuencia se elabora basado en el árbol de descomposición de procesos de para obtener el área de un rectángulo. Debe observarse que la aplicación de la regla uno (secuencia) determina que la escritura del diagrama de secuencia corresponde al árbol de descomposición en forma invertida. Este proceso se repite para los diagramas de secuencia de área del círculo y para el total de metros cuadrados obteniéndose tres diagramas de secuencia básicos.

3er paso.- Se usa la regla doce (combinación) en su modalidad de fusión y se combinan los dos primeros diagramas de secuencia teniendo ahora en un único diagrama el cálculo tanto de las dos áreas.

4° paso.- Se aplica de nuevo la regla doce (combinación) ahora en su modalidad de concatenación para combinar el diagrama obtenido en el paso tres con el último diagrama del paso dos que corresponde al cálculo del total de metros cuadrados. El orden se obtiene al basarse en la regla nueve (existencia) ya que para calcular tmc primero se deben obtener las áreas correspondientes.

5° paso.- Se realiza el primer refinamiento al aplicar la regla cuatro (colaboración) las salidas del diagrama obtenido en el paso tres corresponden con las entradas el diagrama del total de metros cuadrados por lo que las salidas de uno se vuelven entradas del otro. Un segundo refinamiento podría ser combinar los dos bloques de proceso en uno solo para simplificar el diagrama. Esto no se muestra en la figura.

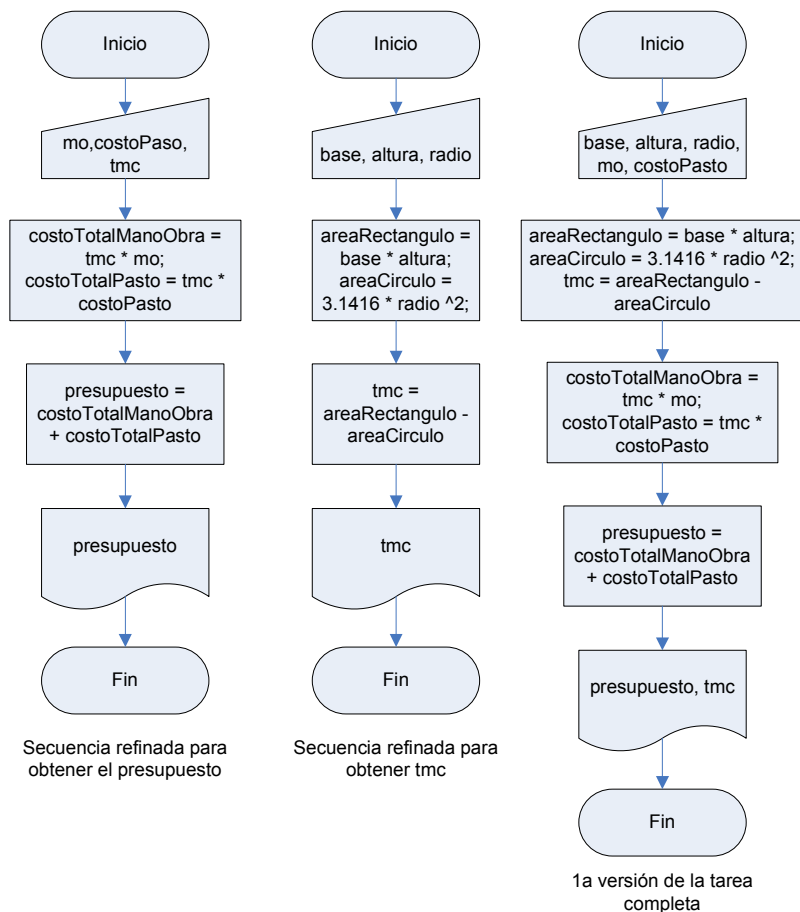


Figura 52. 1ª versión de El Jardín

Esta misma secuencia se hace para la salida de presupuesto obteniéndose el primer diagrama de la figura 52. En la misma figura se observa como se obtiene la 1er versión de solución para el problema El Jardín. Los dos primeros diagramas son las versiones refinadas de cada una de las dos salidas. Aplicando la regla doce (combinación) en su modalidad de concatenación se obtiene el diagrama de la 1ª versión. Debe observarse que se están aplicando también las reglas número tres (composición) dejando en varios cálculos simples el cálculo completo; la número cuatro (colaboración) al eliminar de la entrada a tmc ya que es salida del diagrama de secuencia previo; la número nueve (existencia) al elegir como primer diagrama el que obtiene tmc y como segundo el que obtiene el presupuesto para permitir que las variables involucradas en el cálculo tengan valor antes de utilizarse; y finalmente la número siete (integridad) ya que se ha respetado en todo momento el diseño de la estructura de control.

El diseño de condiciones y de repeticiones se posterga a la segunda iteración por lo que a continuación se prosigue al diseño de pruebas.

3.4.7.2.2.2. Diseño de Pruebas

En materia de diseño de pruebas, solo se puede aplicar una cuyos datos de entrada sean todos válidos. Esto es debido a que en esta primera iteración aún no se han agregado elementos de restricciones y repeticiones para poder incluir casos de prueba sobre estos particulares, únicamente hay diseño de secuencias y esto es lo que debe validarse.

El diseño del caso de prueba queda de la siguiente forma:

CASO 1 Caja Negra	Validar que el cálculo del área de un jardín que se va empastar y su presupuesto sea correcto	
ENTRADAS: base = 20 altura = 10 radio = 2 mo = 25 costoPasto = 50		SALIDAS: tmc = 187.43 presupuesto = 14,057.52

3.4.7.2.3. Fase de Construcción

En esta fase se transcribe el diseño obtenido en código 3GL según las recomendaciones de escritura por esqueletos que se explicó en la metodología. El código resultante del diseño de la primera iteración escrito en C# es el siguiente:

```
using System;

namespace JardinPresupuesto
{
    /// <summary>
    /// version 0.1 del problema El Jardin
    /// se omiten acentos por compatibilidad
    /// </summary>
    class Class1
    {
        /// <summary>
        /// Punto de entrada principal de la aplicacion.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            // se definen las variables de acuerdo al diccionario de datos
            double presupuesto, tmc, mo, areaRectangulo, areaCirculo,
                costoTotalManoObra, costoTotalPasto, lado, altura, radio,
                costoPasto;

            // inicio
            // se realiza la lectura de las variables tal como lo indica el diseño
            // agregando la impresion de las preguntas para facilidad del usuario
            Console.Write("Dime la base: ");
            lado = System.Double.Parse(Console.ReadLine());

            Console.Write("Dime la altura: ");
            altura = System.Double.Parse(Console.ReadLine());

            Console.Write("Dime el valor de el radio: ");
            radio = System.Double.Parse(Console.ReadLine());
        }
    }
}
```

```
Console.Write("Dime el costo de la mano de obra ");
mo = System.Double.Parse(Console.ReadLine());

Console.Write("Dime el costo del pasto ");
costoPasto=System.Double.Parse(Console.ReadLine());

// se realizan los calculos segun se indica en el diseño
areaRectangulo = lado * altura;
areaCirculo = 3.1416 * radio * radio;
tmc = areaRectangulo - areaCirculo;

costoTotalPasto = tmc * costoPasto;
costoTotalManoObra = tmc * mo;
presupuesto = costoTotalManoObra + costoTotalPasto;

// se emiten las salidas
Console.WriteLine("El total de Metros cuadrados de pasto es:
    {0}", tmc);
Console.WriteLine("El presupuesto total es: {0}", presupuesto);

// esta sentencia solo es para detener la ejecucion y poder ver
// el resultado que se obtuvo
Console.ReadLine();
// fin del programa
}
}
}
```

3.4.7.2.4. Fase de Pruebas

Es esta fase se aplican las pruebas según el diseño elaborado. En esta primera iteración únicamente se tiene una prueba diseñada que corresponde a evaluar si los cálculos están bien realizados y se generan ambos resultados correctos. En la figura 53 se muestra la pantalla en la que se ejecutó el caso de prueba y cuyo resultado es correcto. Si no lo hubiera sido se realizarían los pasos de revisión propuestos en la metodología.

3.4.7.2.5. Fase de Liberación

En esta etapa se integran todos los productos obtenidos de cada fase en un documento final y se le denomina versión 0.1.

```

C:\Documents and Settings\Ricardo Vargas\Mis documentos\Visual Studio Projects\JardinVe...
Dine la base: 20
Dine la altura: 10
Dine el valor de el radio: 2
Dine el costo de la mano de obra 25
Dine el costo del pasto 50
El total de Metros cuadrados de pasto es: 187.4336
El presupuesto total es: 14057.52
    
```

Figura 53. Caso de prueba 1. Validación de secuencia

3.4.7.3. 2ª Iteración

Durante la segunda iteración se agregan nuevos elementos en cada una de las fases. En la fase de análisis se agrega la identificación de restricciones; en la fase de diseño se agregan los diseños de restricciones y los diseños de pruebas correspondientes; en la fase de construcción se agrega al código los elementos identificados y diseñados; en la fase de pruebas se aplican las pruebas diseñadas para validar si su liberación es posible. Todo esto conduce a tener una segunda versión de la solución al problema el Jardín.

3.4.7.3.1. Fase de Análisis

En la primera iteración se realizaron las correspondientes identificaciones de salidas, procesos y entradas. Ese análisis se conserva en esta segunda iteración y se ve enriquecido al agregar la identificación de restricciones.

3.4.7.3.1.1. Identificación de Restricciones

De acuerdo a la metodología, se pueden identificar dos tipos de restricciones: las del dominio del problema (mundo real) y las del dominio de la solución (del mundo virtual). En este caso de estudio únicamente se tienen restricciones del mundo real. Las restricciones se escriben en forma de condiciones que definen el dominio al que pertenece cada dato.

El primer paso es definir el dominio de cada dato. Esto se hace utilizando el diccionario de datos y agregando la columna de dominio a cada uno de ellos (ver tabla 15). A continuación se citan los dominios de cada dato identificado:

Nombre	Descripción	Tipo	Categoría	Dominio
presupuesto	Presupuesto total de sembrar pasto en el jardín	Real	Salida	Presupuesto > 0 (mayor que cero)
tmc	Total de metros cuadrados de pasto requeridos	Real	Salida	tmc > 0 (mayor que cero)
mo	Mano de Obra de colocación por metro	Real	Entrada	mo > 0 (mayor que cero)
areaRectangulo	Total de metros cuadrados del rectángulo en donde se sembrará el pasto	Real	Proceso	areaRectangulo > 0 (mayor que cero)
areaCirculo	Total de metros cuadrados	Real	Proceso	$0 < \text{areaCirculo} < \text{areaRectangulo}$ (mayor que cero y menor que areaRectangulo)
costoTotalManoObra	Costo total de la mano de obra necesaria	Real	Proceso	costoTotalManoObra > 0 (mayor que cero)
costoTotalPasto	Costo tota de los metros cuadrados de pasto a comprar	Real	Proceso	costoTotalPasto > 0 (mayor que cero)
base	Lado más largo del rectángulo del terreno dónde se sembrará el jardín	Real	Entrada	base > 0 (mayor que cero)
altura	Lado más corto del terreno dónde se sembrará el jardín	Real	Entrada	altura > 0 (mayor que cero)
radio	Radio de la fuente central	Real	Entrada	radio > 0 (mayor que cero) radio < (altura/2) (menor que la mitad de la altura)
costoPasto	Costo del metro cuadrado de pasto	Real	Entrada	costoPasto > 0 (mayor que cero)

Tabla 15. Diccionario de datos 2a versión

El siguiente paso es tomar las variables de entrada y especificar que sus dominios sean definidos como condiciones de control para procesos de repetición que se analizarán en la tercera iteración.

Condiciones de Control para ciclos abiertos:

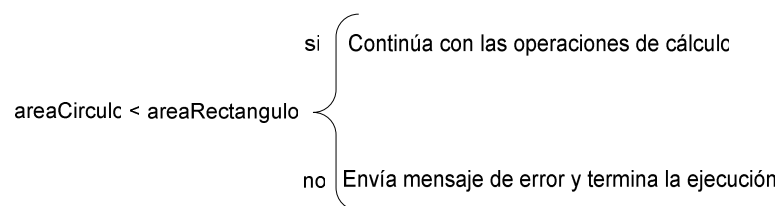
- Variable: mo
Identificación de restricción: ciclo de lectura abierto
Condición de control: mo > 0
- Variable: base
Identificación de restricción: ciclo de lectura abierto

Condición de control: $base > 0$

- Variable: altura
Identificación de restricción: ciclo de lectura abierto
Condición de control: $altura > 0$
- Variable: radio
Identificación de restricción: ciclo de lectura abierto
Condición de control: $radio > 0$ AND $radio < (altura/2)$
- Variable: costoPasto
Identificación de restricción: ciclo de lectura abierto
Condición de control: $costoPasto > 0$

El siguiente paso es tomar las variables de proceso y comprobar si la definición de su dominio definido queda satisfecha por validaciones de entrada o si son necesarias estructuras condicionales con acciones para cada una de los posibles casos de verdadero o falso. En este segundo caso se puede utilizar cualquiera de las sintaxis explicadas en la sección 3.4.2.4.

- Variable de proceso: areaRectangulo
Identificación de restricción: No hay. Su dominio queda satisfecho al validar los datos de entrada base y altura.
- Variable de proceso: areaCirculo
Identificación de restricción: Debe ser mayor que areaRectangulo
Definición de condición:



- Variable de proceso: costoTotalManoObra
Identificación de restricción: No hay. Su dominio queda satisfecho al validar el dato de entrada mo y los necesarios para calcular tmc.
- Variable de proceso: costoTotalPasto
Identificación de restricción: No hay. Su dominio queda satisfecho al validar el dato de entrada costoPasto y los necesarios para calcular tmc.

El último paso es tomar las variables de salida y evaluar si la satisfacción de su dominio requiere de alguna condición adicional o si queda cubierto a través de las validaciones previas.

- Variable de proceso: presupuesto
Identificación de restricción: No hay. Su dominio queda satisfecho al validar los datos de entrada base, altura, radio, mo y costoPasto
- Variable de proceso: tmc
Identificación de restricción: No hay. Su dominio queda satisfecho al validar los datos de entrada base, altura y radio

3.4.7.3.2. Fase de Diseño

En esta iteración solamente se han agregado las restricciones que se diseñarán a través de estructuras de control de tipo condicional. A pesar de ya hay identificado condiciones para ciclos será hasta la tercera iteración en que se agregarán. Por lo tanto solamente se agregarán diseño de restricciones y de pruebas de esas restricciones.

3.4.7.3.2.1. Diseño de Restricciones

De acuerdo a la metodología de microingeniería de software se diseña un procedimiento para la restricción identificada en el análisis y posteriormente, aplicando las reglas de la lógica (heurísticas) se combinara este procedimiento con la 1ª versión que contiene ya una tarea completa y funcional.

Tomando la información del análisis se elabora el procedimiento partiendo de una estructura condicional. Se elige la estructura if-then-else debido a que el análisis identificó acciones tanto para la evaluación verdadera como para la falsa. La figura 54 muestra el diseño de este procedimiento.

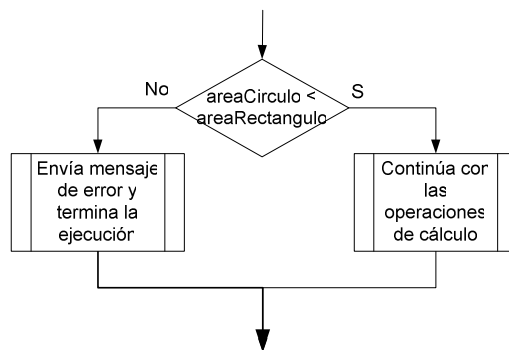


Figura 54. Diseño de procedimiento condicional

Integración y refinamiento utilizando las reglas de la lógica. En la figura 55 se puede observar este proceso. Al extremo izquierdo se tiene la versión de la tarea al final de la 1ª iteración, luego le sigue el procedimiento diseñado.

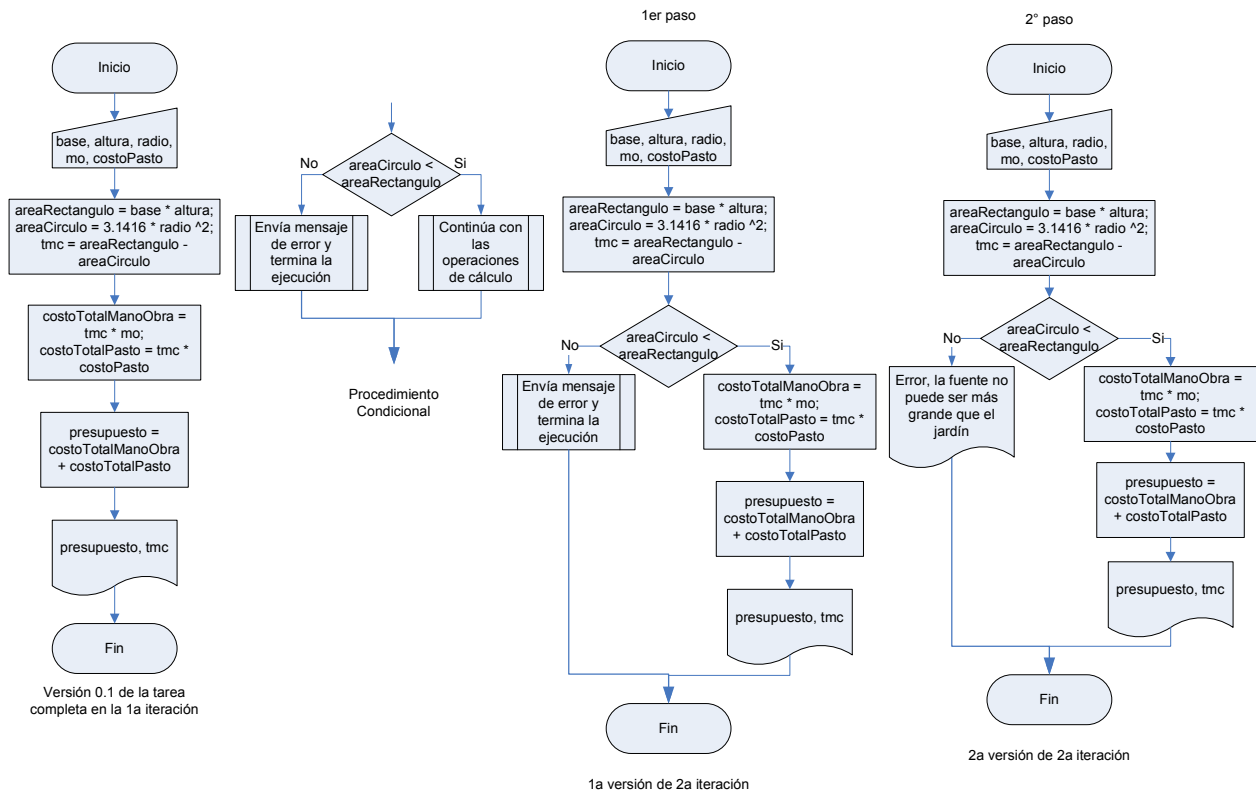


Figura 55. Integración y refinamiento para 2a iteración

El 1er paso es agregar el procedimiento condicional a la tarea. El punto adecuado para su inserción es justo después de que las variables que intervienen en la condición ya tienen valor (aplicando la regla seis -creación-, cinco -cerradura- y doce -combinación- en su modalidad de anidación), eso se logra en seguida del primer símbolo de proceso. Como el diseño del procedimiento indicaba que en el caso de que la condición sea evaluada como verdadera se debe continuar con el resto de las operaciones así que aplicando la regla cinco (cerradura) se toma el resto de las operaciones y se colocan en el lugar de la estructura del lado verdadera del procedimiento condicional. Es importante vigilar la aplicación de la regla siete (integridad) y diagramar la estructura completa tal y como su diseño lo marca conservando su lado de evaluación falsa con la definición de una estructura para enviar un mensaje de error.

En el segundo paso se sustituye, aplicando la regla cinco (cerradura), la estructura definida para el mensaje de error por las acciones adecuadas para cumplir este procedimiento. Con esto se obtiene una segunda versión completa para la segunda iteración.

3.4.7.3.2.2. Diseño de Pruebas

En esta iteración se agregan pruebas para la condición. Se consideran dos casos según cita la metodología: un caso para cuando $areaCirculo$ es menor que $areaRectangulo$; y otro para cuando $areaCirculo$ es mayor o igual que $areaRectangulo$;

CASO 2 Caja Negra	Validar el caso de cuando $areaCirculo$ es menor que $areaRectangulo$ verdadera la condición sea evaluada como verdadera
ENTRADAS: base = 10 altura = 5 radio = 1 mo = 20 costoPasto = 25	SALIDAS: Ejecuta los cálculos y muestra los resultados: tmc = 46.85 presupuesto = 2,108.62

CASO 3 Caja Negra	Validar el caso de cuando $areaCirculo$ es mayor que $areaRectangulo$ verdadera la condición sea evaluada como falsa
ENTRADAS: base = 10 altura = 5 radio = 7 mo = 25 costoPasto = 50	SALIDAS: Mensaje de error: El área de la fuente no puede ser mayor que el jardín

3.4.7.3.3. Fase de Construcción

En esta fase se modifica la versión 0.1 para obtener la versión 0.2 incluyendo la condición definida en el nuevo diseño.

```
using System;

namespace JardinPresupuesto
{
    /// <summary>
    /// version 0.2 del problema El Jardin
    /// se omiten los acentos por compatibilidad
    /// </summary>
    class Class1
    {
        /// <summary>
        /// Punto de entrada principal de la aplicacion.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            // se definen las variables de acuerdo al diccionario de datos
            double presupuesto, tmc, mo, areaRectangulo, areaCirculo,
                costoTotalManoObra, costoTotalPasto, lado, altura, radio,
                costoPasto;

            // inicio
            // se realiza la lectura de las variables tal como lo indica el diseño
            // agregando la impresion de las preguntas para facilidad del usuario
            Console.Write("Dime la base: ");
            lado = System.Double.Parse(Console.ReadLine());
```



```
Console.Write("Dime la altura: ");
altura = System.Double.Parse(Console.ReadLine());

Console.Write("Dime el valor de el radio: ");
radio = System.Double.Parse(Console.ReadLine());

Console.Write("Dime el costo de la mano de obra ");
mo = System.Double.Parse(Console.ReadLine());

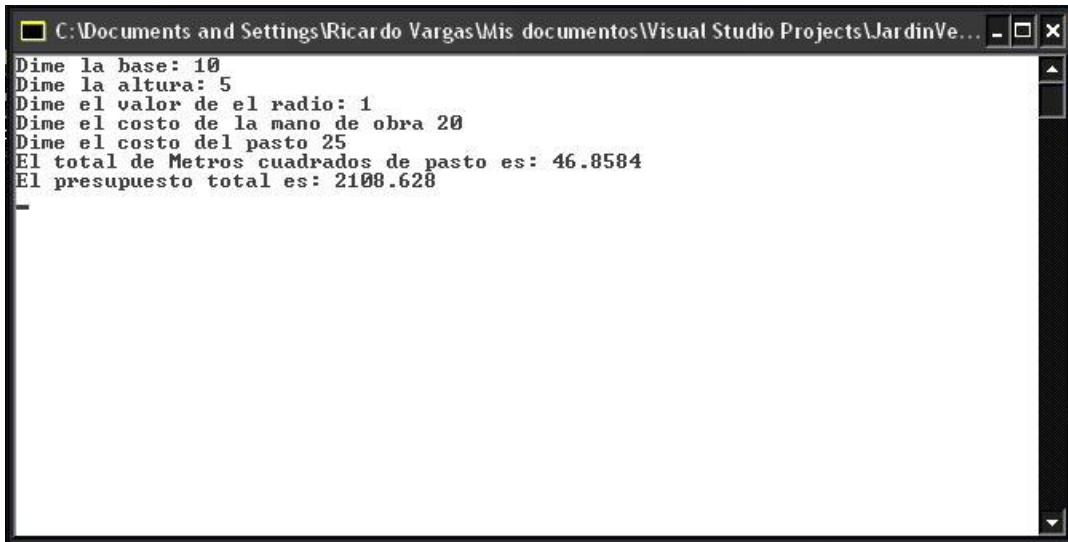
Console.Write("Dime el costo del pasto ");
costoPasto=System.Double.Parse(Console.ReadLine());

// se realizan los calculos segun se indica en el diseño
areaRectangulo = lado * altura;
areaCirculo = 3.1416 * radio * radio;
tmc = areaRectangulo - areaCirculo;

// se aplica la condicion definida
if (areaCirculo < areaRectangulo)
{
    // en caso de que la condicion sea verdadera se ejecuta este bloque
    costoTotalPasto = tmc * costoPasto;
    costoTotalManoObra = tmc * mo;
    presupuesto = costoTotalManoObra + costoTotalPasto;

    // se emiten las salidas
    Console.WriteLine("El total de Metros cuadrados de pasto es:
        {0}",tmc);
    Console.WriteLine("El presupuesto total es: {0}",presupuesto);

    // esta sentencia solo es para detener la ejecucion y poder ver
    // el resultado que se obtuvo
    Console.ReadLine();
    // fin del programa
}
else
{
    // en caso de que la condicion sea falsa se ejecuta este bloque
    Console.WriteLine("El área de la fuente no puede ser mayor que el
        jardin");
}
}
}
```

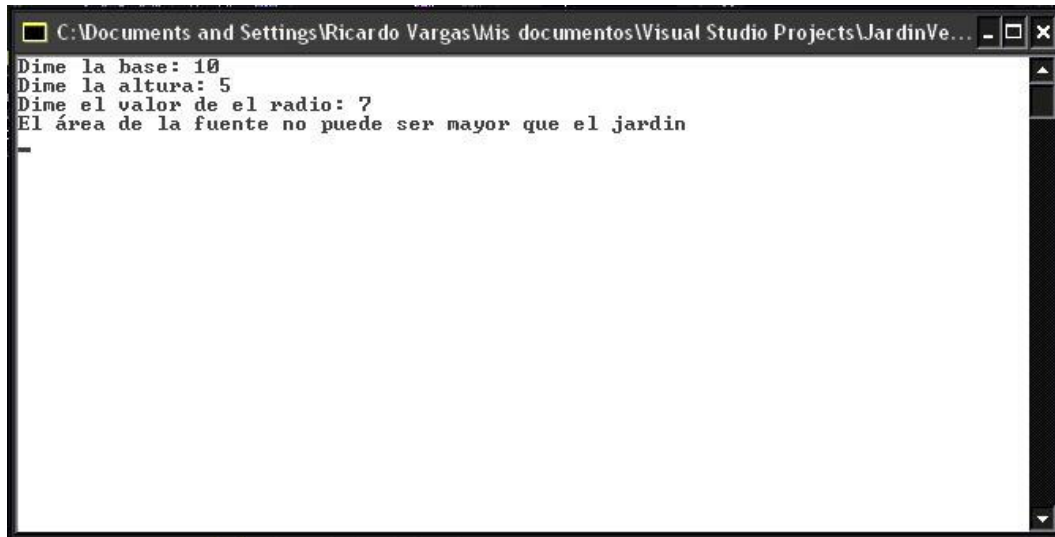


```
C:\Documents and Settings\Ricardo Vargas\Mis documentos\Visual Studio Projects\JardinVe...
Dime la base: 10
Dime la altura: 5
Dime el valor de el radio: 1
Dime el costo de la mano de obra 20
Dime el costo del pasto 25
El total de Metros cuadrados de pasto es: 46.8584
El presupuesto total es: 2108.628
```

Figura 56. Caso de prueba 2 de 2a iteración

3.4.7.3.4. Fase de Pruebas

Se aplican los casos de prueba dos y tres como se pueden ver en las figuras 56 y 57.



```
C:\Documents and Settings\Ricardo Vargas\Mis documentos\Visual Studio Projects\JardinVe...
Dime la base: 10
Dime la altura: 5
Dime el valor de el radio: 7
El área de la fuente no puede ser mayor que el jardin
```

Figura 57. Caso de prueba 3 de 2a iteración

3.4.7.3.5. Fase de Liberación

En esta etapa se integran todos los productos obtenidos de cada fase de esta segunda iteración en un documento final y se le denomina versión 0.2.

3.4.7.4. 3ª Iteración

Durante cada iteración se agregan nuevos elementos a cada una de las fases. Durante esta tercera iteración se considerará lo siguiente: En la fase de análisis se agrega la identificación de repeticiones; en la fase de diseño se agregan los diseños de repeticiones y los diseños de pruebas correspondientes; en la fase de construcción se agrega al código los elementos identificados y diseñados; en la fase de pruebas se aplican las pruebas diseñadas para validar si su liberación es posible. Todo esto conduce a tener una tercera versión de la solución al problema El Jardín.

3.4.7.4.1. Fase de Análisis

Durante esta iteración se agrega la identificación de repeticiones. De acuerdo a la metodología de microingeniería se pueden considerar repeticiones por procesos de validaciones o por necesidades fijadas por el planteamiento del problema. En este caso de estudio únicamente se tienen repeticiones para realizar las lecturas de las variables.

3.4.7.3.4.1. Identificación de Repeticiones

Durante la segunda iteración, en la etapa de identificar condiciones se identificaron cinco variables que deben ser leídas y validadas. Se toman y simplemente se realiza la redacción de cada caso:

- Repetir la lectura de la variable: base
Condición de control para considerar bueno el valor de la lectura: $base > 0$
Condición de control para considerar malo el valor de lectura: $base \leq 0$
En caso de no cumplir la condición mostrar mensaje de error y repetir la lectura
- Repetir la lectura de la variable: altura
Condición de control para considerar bueno el valor de la lectura: $altura > 0$
Condición de control para considerar malo el valor de lectura: $altura \leq 0$
En caso de no cumplir la condición mostrar mensaje de error y repetir la lectura
- Repetir la lectura de la variable: radio
Condición de control para considerar bueno el valor de la lectura: $radio > 0$ AND $radio < (altura/2)$
Condición de control para considerar malo el valor de lectura: $radio \leq 0$ OR $radio \geq (altura/2)$
En caso de no cumplir la condición mostrar mensaje de error y repetir la lectura
- Repetir la lectura de la variable: mo
Condición de control para considerar bueno el valor de la lectura: $mo > 0$
Condición de control para considerar malo el valor de lectura: $mo \leq 0$
En caso de no cumplir la condición mostrar mensaje de error y repetir la lectura
- Repetir la lectura de la variable: costoPasto

Condición de control para considerar bueno el valor de la lectura: $\text{costoPasto} > 0$
Condición de control para considerar malo el valor de lectura: $\text{costoPasto} \leq 0$
En caso de no cumplir la condición mostrar mensaje de error y repetir la lectura

3.4.7.4.2. Fase de Diseño

Durante esta iteración, la fase de diseño agrega el diseño de repeticiones y el de nuevas pruebas, éstas para validar las repeticiones diseñadas.

3.4.7.4.2.1. Diseño de Repeticiones

Cada una de las variables identificadas en el análisis que deben ser repetidas sus lecturas pasa por su etapa de diseño. De acuerdo con la metodología, se debe de elegir el tipo de repetición para cada variable de acuerdo a los siguientes criterios:

1. Si se conoce de antemano el número de repeticiones que se realizarán se elige un ciclo cerrado.
2. Si se desconoce el número de repeticiones que se realizarán se elige un ciclo abierto.
3. Si se trata de un ciclo abierto y se desea que siempre se ejecute al menos una ocasión el cuerpo del ciclo (es decir una o más veces) se elige un ciclo tipo until con la condición al final del ciclo.
4. Si se trata de un ciclo abierto y se desea que el cuerpo del ciclo pueda ser ejecutado cero, una o más veces se elige un ciclo de tipo while con la condición al inicio del ciclo.

Tomando como ejemplo la primera variable (el caso será igual para todas ellas ya que todas son repeticiones para validación por lectura de datos), se realizan las siguientes reflexiones basadas en los criterios antes citados:

1. ¿Se conoce de antemano cuantas veces será necesario preguntar el valor de la base del terreno (base) al usuario en caso de que se equivoque al escribirlo? No, no se sabe, por lo tanto la primera decisión es que se trata de un ciclo abierto.
2. ¿Se desea que se ejecute al menos en una ocasión el cuerpo del ciclo? Si se considera que cada repetición implica la impresión de un mensaje de error entonces la respuesta es No, no se desea imprimir al menos en una ocasión el mensaje de error puesto que la primera lectura puede ser correcta y no hay necesidad de repetirla. Por lo tanto la conclusión es que se utilizará un ciclo abierto del tipo while con la condición al principio.

Con estas decisiones tomadas se inicia el proceso de diseño de restricciones siguiendo los mismos pasos que en la segunda iteraciones. Se elabora el procedimiento para una de las variables y luego se fusiona con la versión 0.2 de la tarea utilizando la regla doce (Combinación) y la cinco (cerradura).

La figura 58 muestra el proceso de integración y refinamiento de una de las variables. El primer diagrama del lado izquierdo contiene la versión 0.2 resultado de la 2ª iteración. El siguiente diagrama es el diagrama de procedimiento elaborado a partir de la estructura de control de ciclo abierto tipo while completándose con la información resultado del análisis.

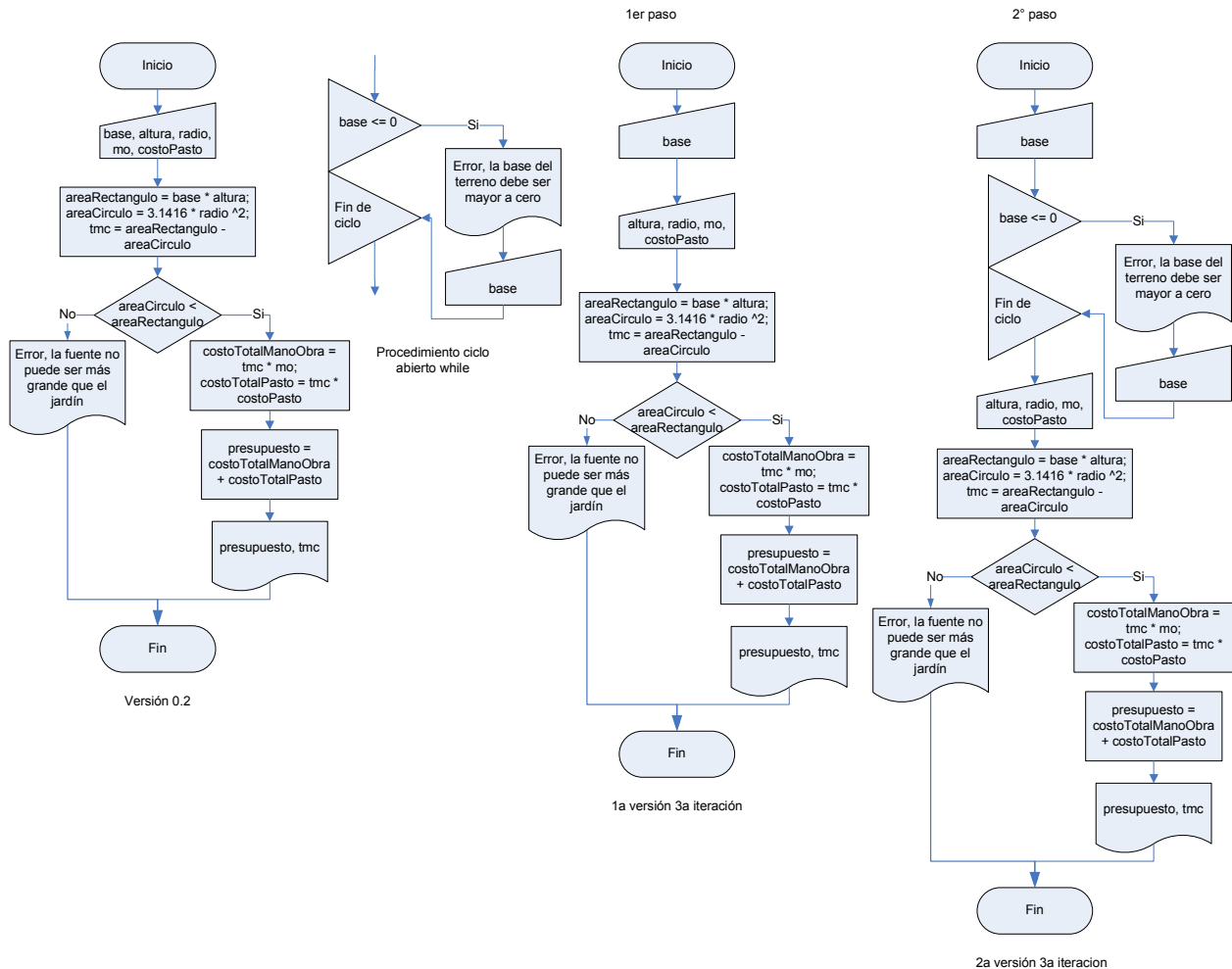


Figura 58. Integración y refinamiento 3a iteración

El primer paso de la integración y refinamiento consiste en aplicar la regla 3 (composición) y dividir la lectura de las variables de entrada en dos; una acción de lectura exclusiva para la base y la segunda para el resto de las variables.

El segundo paso consiste en insertar el procedimiento de lectura de la base en su lugar. Aplicando la regla 6 (creación) y la doce (combinación), en su modalidad de concatenación, se decide colocarla siguiendo el mismo criterio que el explicado en la segunda iteración para colocar las condiciones (además de aplicar la regla 9 – existencia -): Inmediatamente después de la lectura de su valor, por tanto se ubica después de la lectura de la base.

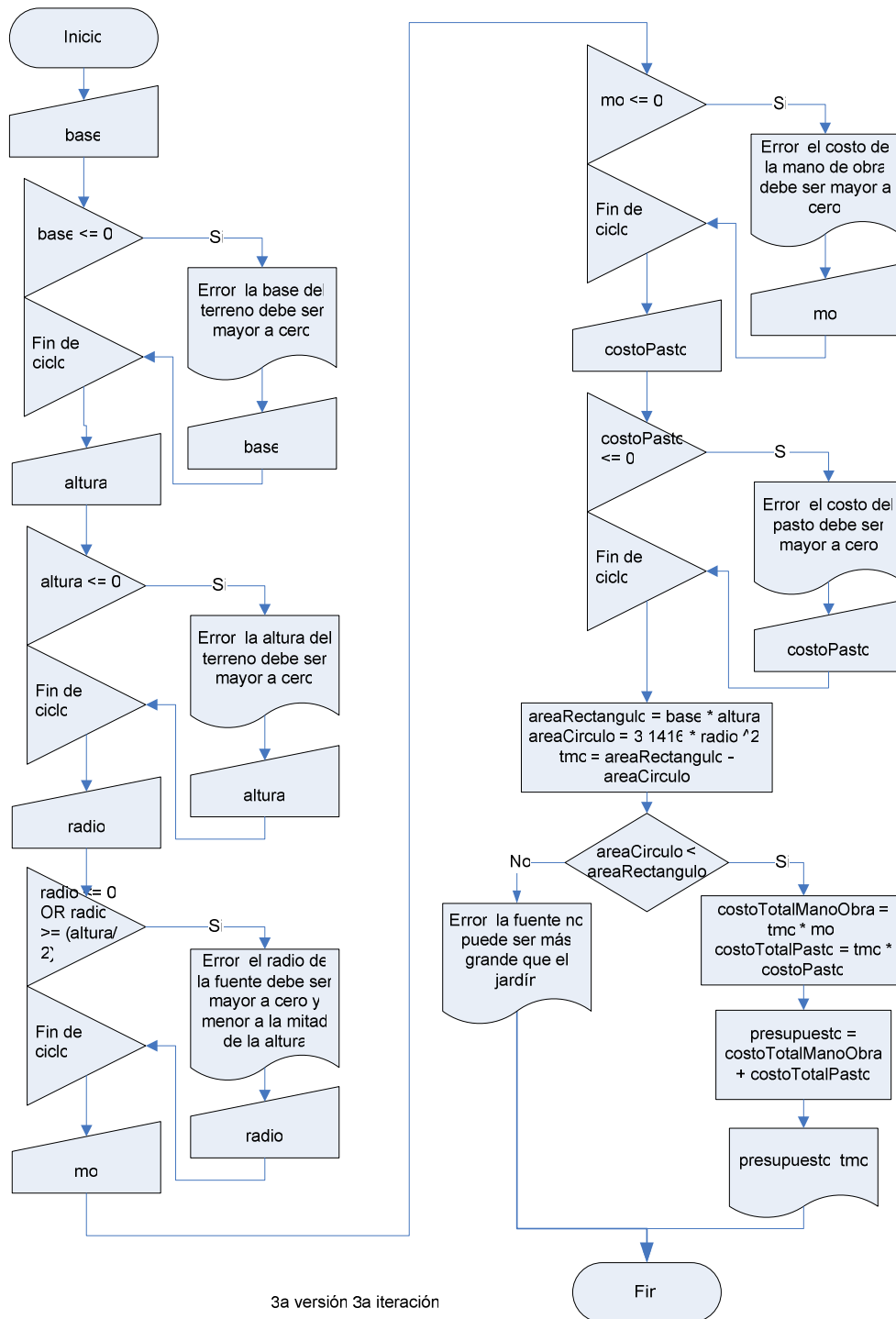


Figura 59. Diagrama final del problema El Jardín

Esta actividad se repite para todas y cada una de las variables que debe ser repetida su lectura mientras haya errores de captura. En la figura 59 se muestra la versión final de esta iteración.

3.4.7.4.2.2. Diseño de Pruebas

De acuerdo con la Metodología de Microingeniería, los ciclos abiertos tipo while se prueban con tres casos: ninguna ejecución del ciclo, una ejecución del ciclo, más de una ejecución.

Los casos de prueba son los siguientes:

CASO 4 Caja Negra	Validar la repetición de lectura de la variable base para el caso en que no debe ejecutarse ni siquiera una vez.	
ENTRADAS: base = 12 altura = 3 radio = 1.5 mo = 15 costoPasto = 35	SALIDAS: tmc = 28.93 presupuesto = 1,446.57	

CASO 5 Caja Negra	Validar la repetición de lectura de la variable base para el caso en que se repite una sola vez.	
ENTRADAS: base = 0 base = 8 altura = 3 radio = 1 mo = 15 costoPasto = 15	SALIDAS: Mensaje de error: La base del rectángulo debe ser mayor a cero tmc = 20.85 presupuesto = 625.75	

CASO 6 Caja Negra	Validar la repetición de lectura de la variable base para el caso en que se repite varias veces.	
ENTRADAS: base = -3 base = -100 base = 13 altura = 5 radio = 2 mo = 25 costoPasto = 50	SALIDAS: Mensaje de error: La base del rectángulo debe ser mayor a cero Mensaje de error: La base del rectángulo debe ser mayor a cero tmc = 52.43 presupuesto = 3,932.52	

3.4.7.4.3. Fase de Construcción

La codificación del diagrama de la figura 59 es la siguiente:

```
using System;
```

```
namespace JardinPresupuesto
{
    /// <summary>
    /// version 0.3 del problema El Jardin
    /// se omiten los acentos por compatibilidad
    /// </summary>
    class Class1
    {
        /// <summary>
        /// Punto de entrada principal de la aplicacion.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            // se definen las variables de acuerdo al diccionario de datos
            double presupuesto, tmc, mo, areaRectangulo, areaCirculo,
                costoTotalManoObra, costoTotalPasto, lado, altura, radio,
                costoPasto;

            // inicio

            // se realiza la lectura de las variables tal como lo indica el diseño
            // agregando la impresion de las preguntas para facilidad del usuario
            Console.WriteLine("Dime la base: ");
            lado = System.Double.Parse(Console.ReadLine());
            while(lado<=0)
            {
                Console.WriteLine("La base del rectángulo debe ser mayor a
                    cero");
                Console.WriteLine("Dime la base: ");
                lado = System.Double.Parse(Console.ReadLine());
            }

            Console.WriteLine("Dime la altura: ");
            altura = System.Double.Parse(Console.ReadLine());
            while (altura <=0)
            {
                Console.WriteLine("La altura del rectángulo debe ser mayor a
                    cero");
                Console.WriteLine("Dime la altura: ");
                altura = System.Double.Parse(Console.ReadLine());
            }

            Console.WriteLine("Dime el valor de el radio: ");
            radio = System.Double.Parse(Console.ReadLine());
            while (radio <= 0 || radio >= (altura/2))
            {
                Console.WriteLine("El radio de la fuente debe ser mayor a cero");
                Console.WriteLine("Dime el valor de el radio: ");
                radio = System.Double.Parse(Console.ReadLine());
            }

            Console.WriteLine("Dime el costo de la mano de obra ");
            mo = System.Double.Parse(Console.ReadLine());
            while (mo<=0)
            {
                Console.WriteLine("El costo de la mano de obra debe ser mayor a
                    cero");
            }
        }
    }
}
```



```
    Console.WriteLine("Dime el costo de la mano de obra ");
    mo=System.Double.Parse(Console.ReadLine());
}

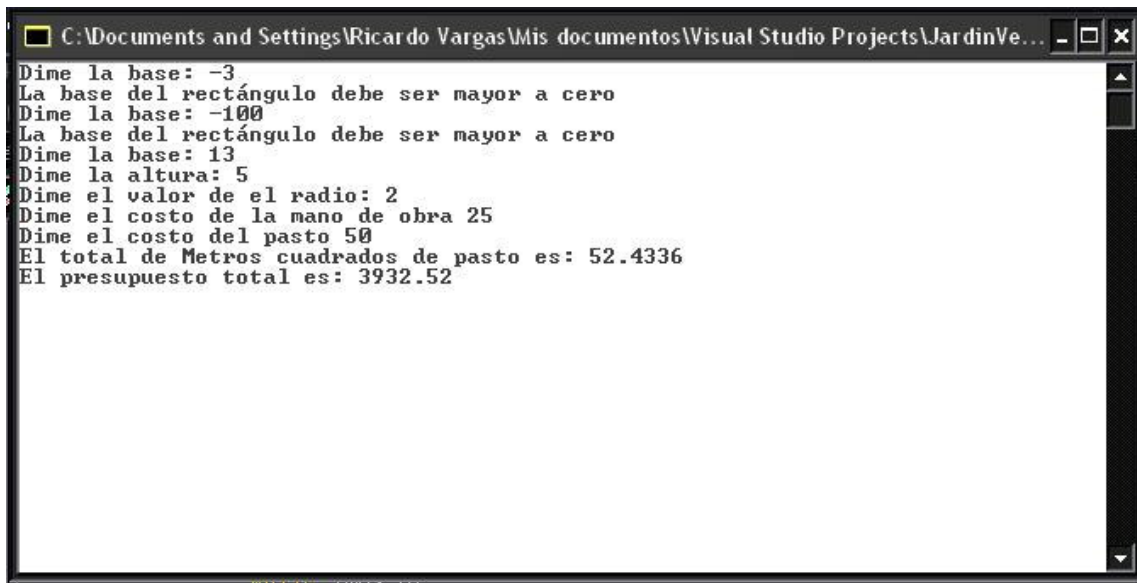
Console.WriteLine("Dime el costo del pasto ");
costoPasto=System.Double.Parse(Console.ReadLine());
while (costoPasto<=0)
{
    Console.WriteLine("El costo del pasto debe ser mayor a cero");
    Console.WriteLine("Dime el costo del pasto ");
    costoPasto=System.Double.Parse(Console.ReadLine());
}

areaRectangulo = lado * altura;
areaCirculo = 3.1416 * radio * radio;

if (areaCirculo < areaRectangulo)
{
    tmc = areaRectangulo - areaCirculo;
    costoTotalPasto = tmc * costoPasto;
    costoTotalManoObra = tmc * mo;
    presupuesto = costoTotalManoObra + costoTotalPasto;
    Console.WriteLine("El total de Metros cuadrados de pasto es:
        {0}",tmc);
    Console.WriteLine("El presupuesto total es: {0}",presupuesto);
}
else
{
    Console.WriteLine("El área de la fuente no puede ser mayor que el
        jardin");
}
Console.ReadLine();
}
}
```

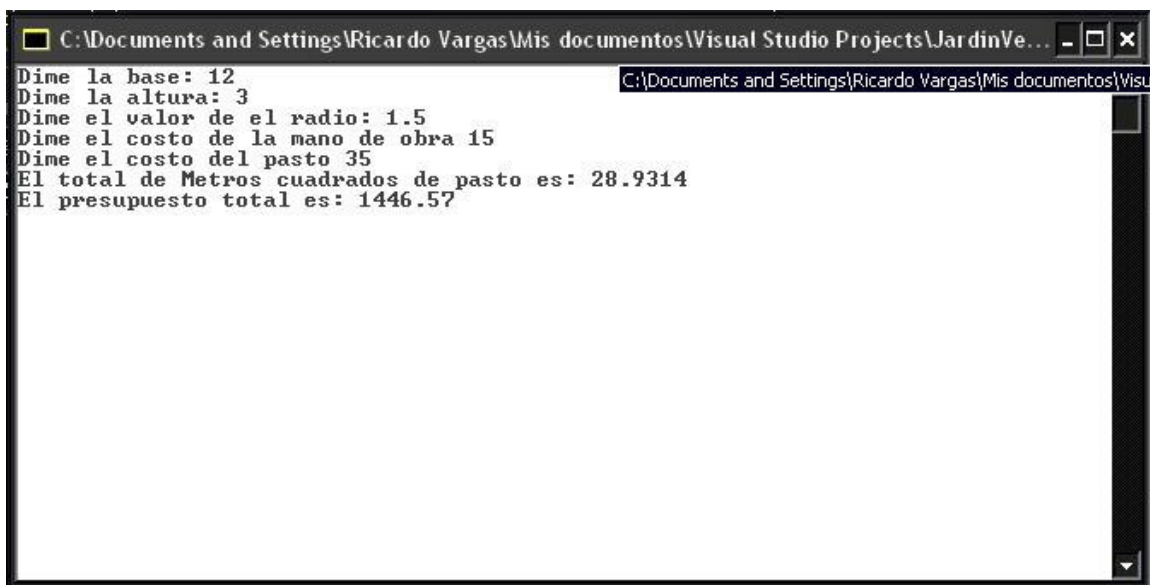
3.4.7.4.4. Fase de Pruebas

La aplicación de los tres nuevos casos de prueba se muestran en las figuras 60. 61 y 62.



```
C:\Documents and Settings\Ricardo Vargas\Mis documentos\Visual Studio Projects\JardinVe...
Dime la base: -3
La base del rectángulo debe ser mayor a cero
Dime la base: -100
La base del rectángulo debe ser mayor a cero
Dime la base: 13
Dime la altura: 5
Dime el valor de el radio: 2
Dime el costo de la mano de obra 25
Dime el costo del pasto 50
El total de Metros cuadrados de pasto es: 52.4336
El presupuesto total es: 3932.52
```

Figura 60. Caso de prueba cuatro de El Jardín



```
C:\Documents and Settings\Ricardo Vargas\Mis documentos\Visual Studio Projects\JardinVe...
Dime la base: 12
Dime la altura: 3
Dime el valor de el radio: 1.5
Dime el costo de la mano de obra 15
Dime el costo del pasto 35
El total de Metros cuadrados de pasto es: 28.9314
El presupuesto total es: 1446.57
```

Figura 61. Caso de prueba cinco de El Jardín

3.4.7.4.5. Fase de Liberación

Al igual que en las iteraciones anteriores se integra toda la documentación resultado de cada una de las fases de esta iteración incluyendo los productos de las iteraciones anteriores y se denomina versión 0.3.

```

C:\Documents and Settings\Ricardo Vargas\Mis documentos\Visual Studio Projects\JardinVe...
Dime la base: 0
La base del rectángulo debe ser mayor a cero
Dime la base: 8
Dime la altura: 3
Dime el valor de el radio: 1
Dime el costo de la mano de obra 15
Dime el costo del pasto 15
El total de Metros cuadrados de pasto es: 20.8584
El presupuesto total es: 625.752
    
```

Figura 62. Caso de prueba seis de El Jardín

3.4.7.5. Producto Final

Una vez concluido todo el proceso se hace un refinamiento final. Esto incluye una revisión de cada una de las fases de la metodología de microingeniería.

3.4.7.5.1. Fase de Análisis

Se integra en un solo documento todos los elementos identificados en cada una de las etapas de esta fase. A continuación se presenta un resumen del ejercicio completo.

+ Identificación de Resultados

Presupuesto para poner pasto en un jardín
 Total de metros cuadrados a Sembrar

+ Identificación de Procesos

Presupuesto = costoTotalManoObra + costoTotalPasto
 costoTotalManoObra = tmc * mo
 costoTotalPasto = tmc * costoPasto
 tmc = areaRectangulo – areaCirculo
 areaRectangulo = base * altura
 areaCirculo = 3.1416 * radio * radio

+ Identificación de Datos de Entrada

Costo de la Mano de Obra por metro cuadrado de colocación de pasto.

Costo del metro cuadrado de pasto

Lado más largo del terreno donde se sembrará el pasto (base)

Lado más corto del terreno donde se sembrará el pasto (lado)

Radio del círculo que forma la fuente que se colocará al centro del terreno.

+ Identificación de restricciones o condiciones

La mano de obra debe ser mayor a 0

El lado más largo (base) del rectángulo del terreno debe ser mayor a 0

El lado más corto (altura) del rectángulo del terreno debe ser mayor a 0

El radio de la fuente debe ser mayor a 0 y menor a la mitad de la altura

El área del rectángulo tiene que ser mayor que el área de la fuente.

+ Repeticiones o iteraciones

Repetir la lectura de los valores de las variables de entrada en caso de que sean incorrectas

+ Diccionario de datos final

Nombre	Descripción	Tipo	Categoría	Dominio
presupuesto	Presupuesto total de sembrar pasto en el jardín	Real	Salida	Presupuesto > 0
tmc	Total de metros cuadrados de pasto requeridos	Real	Salida	tmc > 0
mo	Mano de Obra de colocación por metro	Real	Entrada	mo > 0
areaRectangulo	Total de metros cuadrados del rectángulo en donde se sembrará el pasto	Real	Proceso	areaRectangulo > 0
areaCirculo	Total de metros cuadrados	Real	Proceso	$0 < \text{areaCirculo} < \text{areaRectangulo}$
costoTotalManoObra	Costo total de la mano de obra necesaria	Real	Proceso	costoTotalManoObra > 0
costoTotalPasto	Costo tota de los metros cuadrados de pasto a comprar	Real	Proceso	costoTotalPasto > 0
base	Lado más largo del rectángulo del terreno dónde se sembrará el jardín	Real	Entrada	base > 0
altura	Lado más corto del terreno dónde se sembrará el jardín	Real	Entrada	altura > 0
radio	Radio de la fuente central	Real	Entrada	radio > 0
costoPasto	Costo del metro cuadrado de pasto	Real	Entrada	costoPasto > 0

Tabla 16. Diccionario de datos versión final

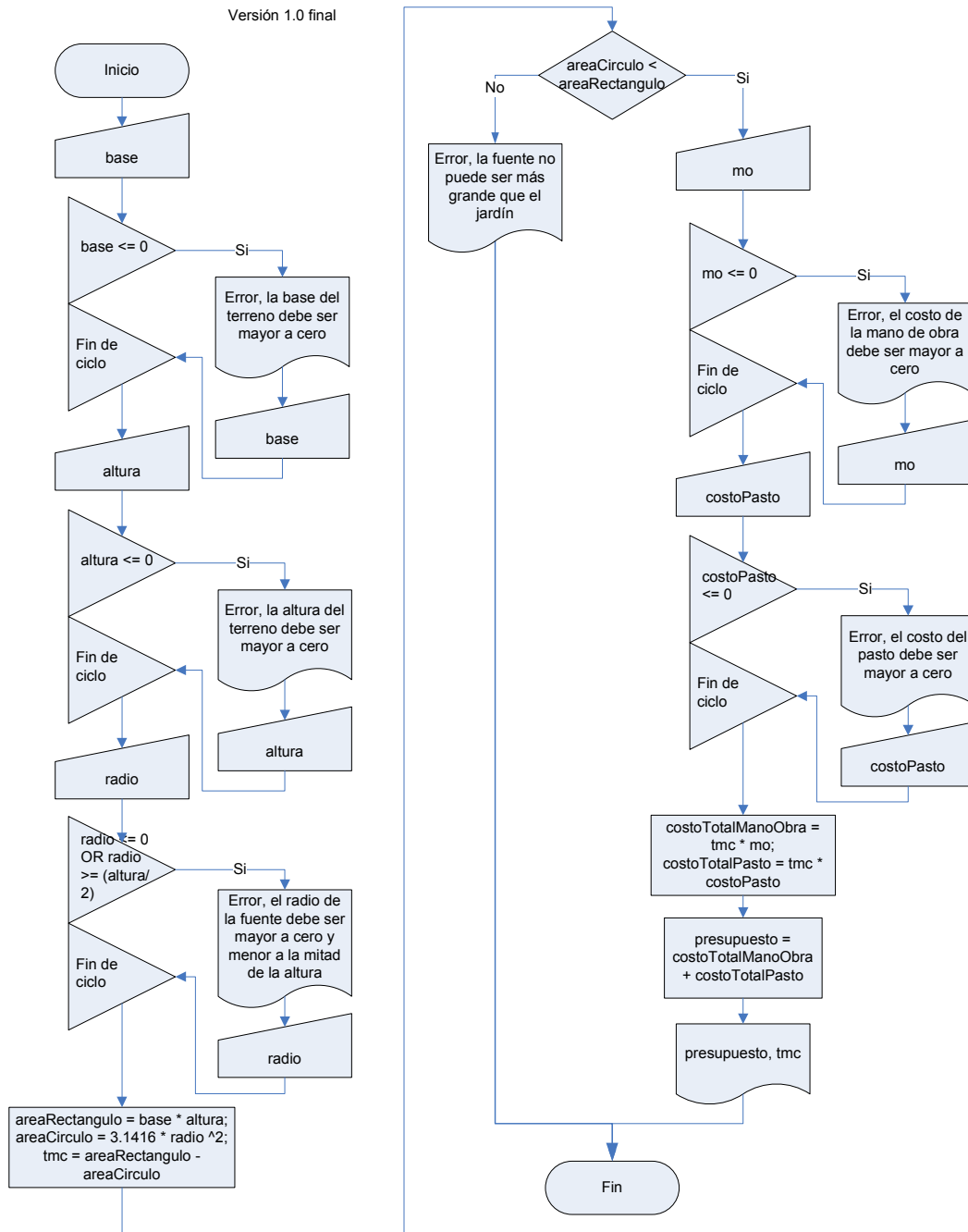


Figura 63. Versión 1.0 final de El Jardín

3.4.7.5.2. Fase de Diseño

Se revisa el diseño integrado de las secuencias, condiciones e iteraciones que forman la tarea y los casos de prueba apropiados para validar y descubrir errores.

En el caso de este ejercicio se puede hacer un proceso de refinamiento final que se muestra en la figura 63 en el cual se ha reubicado la lectura del costo de la mano de obra y costo del pasto después de haber validado que las dimensiones del terreno del jardín sean correctas. Esto permite,

en caso de error, no realizar más operaciones que representan un costo de proceso y únicamente realizarlas en caso de que todo este correcto.

3.4.7.5.3. Fase de Construcción

Partiendo del diseño refinado final se obtiene el siguiente código:

```
using System;

namespace JardinPresupuesto
{
    /// <summary>
    /// version 1.0 del problema El Jardin
    /// se omiten los acentos por compatibilidad
    /// </summary>
    class Class1
    {
        /// <summary>
        /// Punto de entrada principal de la aplicacion.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            // se definen las variables de acuerdo al diccionario de datos
            double presupuesto, tmc, mo, areaRectangulo, areaCirculo,
                costoTotalManoObra, costoTotalPasto, lado, altura, radio,
                costoPasto;

            // inicio

            // se realiza la lectura de las variables tal como lo indica el diseño
            // agregando la impresion de las preguntas para facilidad del usuario
            Console.Write("Dime la base: ");
            lado = System.Double.Parse(Console.ReadLine());
            while (lado<=0)
            {
                Console.WriteLine("La base del rectángulo debe ser mayor a
                    cero");
                Console.Write("Dime la base: ");
                lado = System.Double.Parse(Console.ReadLine());
            }

            Console.Write("Dime la altura: ");
            altura = System.Double.Parse(Console.ReadLine());
            while(altura <=0)
            {
                Console.WriteLine("La altura del rectángulo debe ser mayor a
                    cero");
                Console.Write("Dime la altura: ");
                altura = System.Double.Parse(Console.ReadLine());
            }

            Console.Write("Dime el valor de el radio: ");
            radio = System.Double.Parse(Console.ReadLine());
            while (radio <= 0 || radio >= (altura/2))
```

```
{
    Console.WriteLine("El radio de la fuente debe ser mayor a cero");
    Console.Write("Dime el valor de el radio: ");
    radio = System.Double.Parse(Console.ReadLine());
}

// se calculan los elementos para validar el tamaño del terreno
areaRectangulo = lado * altura;
areaCirculo 3.1416 * radio * radio;

// se valida el terreno
if (areaCirculo < areaRectangulo)
{
    // si el terreno es correcto se continúa con la lectura de las
    // siguientes variables
    Console.Write("Dime el costo de la mano de obra ");
    mo = System.Double.Parse(Console.ReadLine());
    while(mo<=0)
    {
        Console.WriteLine("El costo de la mano de obra debe ser mayor a
        cero");
        Console.Write("Dime el costo de la mano de obra ");
        mo = System.Double.Parse(Console.ReadLine());
    }

    Console.Write("Dime el costo del pasto ");
    costoPasto=System.Double.Parse(Console.ReadLine());
    while(costoPasto<=0)
    {
        Console.WriteLine("El costo del pasto debe ser mayor a cero");
        Console.Write("Dime el costo del pasto ");
        costoPasto=System.Double.Parse(Console.ReadLine());
    }

    // se realizan los calculos finales
    tmc = areaRectangulo - areaCirculo;
    costoTotalPasto = tmc * costoPasto;
    costoTotalManoObra = tmc * mo;
    presupuesto = costoTotalManoObra + costoTotalPasto;
    Console.WriteLine("El total de Metros cuadrados de pasto es:
    {0}",tmc);
    Console.WriteLine("El presupuesto total es: {0}",presupuesto);
}
else
{
    // en caso de error se envia el mensaje correspondiente.
    Console.WriteLine("El área de la fuente no puede ser mayor que el
    jardin");
}
Console.ReadLine();
}
}
```

3.4.7.5.4. Fase de Pruebas

Si por cada variable de entrada en ciclo se preparan tres casos de prueba más los seis que ya se habían definido se obtiene un total 18 casos. Además, se deberán agregar los cuatro requeridos para validar la condición compuesta del radio ya que también tiene que compararse contra el valor de la mitad de la altura ($\text{radio} < 0$ OR $\text{radio} > (\text{altura}/2)$). Esto asciende a 24 casos. Todos ellos se ejecutan con la versión final refinada.

3.4.7.5.5. Fase de Liberación

Si los casos de prueba fueron exitosos se realiza la integración final de toda la documentación para poder liberar la versión 1.0 del producto.

3.5 Dimensión Herramental: Diseño de Algoritmos Asistido por Computadora

Durante la fase de diseño algorítmico es necesario que exista un lenguaje de modelado que facilite este proceso. De acuerdo al reporte del estado del arte, los lenguajes de modelado actuales son los mismos que se elaboraron hace ya años, entre ellos está el pseudocódigo, los diagramas de flujo y los ordinogramas. Esta limitación ocasiona que muchos programadores se salten el proceso de diseño y directamente vallan directamente a la fase de codificación en algún 3GL. Siendo congruentes con el planteamiento de primero hacer un análisis, luego el diseño para poder pasar a la codificación, pruebas y liberación; el proceso de diseño es una parte fundamental en el descubrimiento del algoritmo y debe ser previo a la codificación en lenguajes de programación.

3.5.1. La Programación Visual Imperativa

El ser humano percibe su entorno a través de sus sentidos: la vista, el gusto, el olfato, el tacto y el oído. La combinación de los cinco permite una comprensión del cosmos tal y como se le conoce, la ausencia de alguno limita las capacidades de comprensión y definición del mundo en si mismo. Sin embargo, a pesar de la importancia de todos ellos, el ser humano se ha caracterizado por darle preferencia a su percepción visual. Por ello desde el nacimiento de la computación como área del conocimiento Goldstein y von Neumann en 1947 ya aplicaban los diagramas como representación visual de los algoritmos utilizando cajas, círculos y otras figuras [94].

La representación visual más difundida son los diagramas de flujo, tanto en su versión de flujo de control como de flujo de datos. Su auge inicia a principios de los años 60's y su declive a finales de los 80's aunque hay materiales de estudio como el de Scanlan en 1989 [90], Blackwell en 1996 [10] y Good en 1999 [46] que los mantuvieron vivos y como tema de debate. En fechas recientes se ha desarrollado una nueva oleada de estudios sobre la ahora llamada *Programación Visual* y los lenguajes que la soportan.

Si bien es cierto que la intención de este trabajo es el modelo de construcción de algoritmos apoyado en heurísticas, estos algoritmos deben ser diseñados y modelados por algún lenguaje previo a su codificación en lenguajes 3GL así que se hará una revisión de los nuevos planteamientos de los lenguajes de programación visual.

3.5.1.1. Definición de Lenguaje de Programación Visual

Hasta hace pocos años todo el mundo de la computación se basaba en procesos textuales. Los sistemas operativos estaban basados en comandos de instrucciones, el manejo de las bases de datos se codificaban en lenguajes denominados 4GL, la programación imperativa, lógica y funcional también estaba basada en textos. La preferencia visual del ser humano invitó a los ingenieros de software a crear nuevos productos cuyas interfaces fueran más *amigables*⁷ y éstas se enriquecieron con colores, sonidos, iconos, animaciones y un sin fin de elementos que permiten tener una experiencia de uso más rica aunque no siempre más eficiente.

Algunas de las áreas de la computación se han beneficiado de forma importante con respecto al uso de múltiples medios disponibles para la construcción de interfaces. Por ejemplo, los sistemas operativos ahora todos son gráficos, con animaciones y audio y pareciera que la tendencia es a aumentar aún más la experiencia visual y auditiva de los usuarios; otra área que se ha visto beneficiada de esta tendencia son las bases de datos, las cuales ahora se modelan utilizando imágenes visuales para construir el código de la creación de tablas dominios e índices así como de la construcción simplificada de consultas gracias a las capacidades gráficas de los entornos actuales.

A pesar de este auge, la programación imperativa ha encontrado aún pocos éxitos en su competencia con sus contrapartes textuales aunque las expectativas demandan nuevos servicios que permitan al desarrollador escribir el menor número de líneas de código y que agilicen el tiempo de implantación. La programación visual es una de las promesas de solución.

Marriot y Meyer en [69] definen a un lenguaje visual como un conjunto de diagramas u objetos gráficos con los que pueden componerse sentencias válidas en este lenguaje, donde un diagrama es una colección de símbolos en un espacio de dos o tres dimensiones y cuya característica principal es que al menos en el conjunto de símbolos terminales se define un objeto gráfico

Se entiende por programación visual el uso de expresiones visuales tales como dibujos, gráficas e iconos dentro del proceso de programación, estableciendo formas de comunicación entre éstas, para conducir al programador dentro de un marco de trabajo más identificado con su manera de pensar y relacionar, reduciendo el tiempo y esfuerzo dedicado con respecto a la programación tradicional.

Existen dos enfoques sobre la programación visual [18]:

- Ambientes de programación visual (por sus siglas en inglés VPE)

⁷ El término amigable se ha aplicado con la connotación de facilidad de uso y desató debates sobre que debía ser considerado “amigable” para un ser humano. Hoy el termino usabilidad está más difundido.

- Lenguajes de programación visual (por sus siglas en inglés VPL)

La diferencia entre estas dos perspectivas es que los VPE únicamente proporcionan al usuario medios visuales o gráficos para la creación de sistemas, pero la gramática con la que trabajan es textual. Los VPL en cambio, combinan un ambiente de desarrollo basado en símbolos visuales, incorporando la posición espacial de los objetos y cuentan con una gramática visual en la que por lo menos uno de sus símbolos terminales es un objeto gráfico.

Según Chang en [24], el uso de símbolos gráficos como parte del proceso de comunicación visual tiene dos representaciones: la parte lógica y la parte física; ambas partes deben ser interpretadas para dar forma al lenguaje de aplicación. La parte lógica se refiere a las funciones relaciones, restricciones y comunicaciones ejecutadas por el símbolo; y la parte física al diseño de la imagen. Existen también criterios de evaluación de los lenguajes de programación visual, publicados por Kiper en [60], siendo estos: su naturaleza visual, funcionalidad, facilidad de comprensión, paradigma soportado y escalabilidad.

3.5.1.2. Expectativas sobre la Programación Visual

Por supuesto, las expectativas que se tienen sobre la programación visual son grandes. Se espera que pueda simplificar el proceso de programación, hacerlo más fácil, más legible, más mantenible y en general ser más productivo que usando sus contrapartes textuales.

Joyanes en [56] propone que las ventajas que debe aportar un lenguaje visual (como lo eran los diagramas de flujo) deben ser los siguientes:

1. Rápida comprensión de las relaciones
2. Análisis efectivo de las diferentes secciones del programa.
3. Los diagramas de flujo pueden utilizarse como modelos de trabajo en el diseño de nuevos sistemas
4. Comunicación con el usuario.
5. Documentación adecuada de los programas.
6. Codificación eficaz de los programas
7. Depuración y pruebas ordenadas de programas.

Además de los principios de Kiper (naturaleza visual, funcionalidad, facilidad de comprensión, paradigma soportado y escalabilidad), Fitter en [38] propuso cinco principios para responder a cuando un diagrama es un buen lenguaje. La tabla 17 contiene estos principios.

Principio	Descripción	Aplicación a diagramas de flujo
Relevancia	La información codificada de manera perceptiva, en lugar de simbólica, debe ser relevante.	Dan relevancia al flujo de control, oscureciendo las relaciones lógicas, procesos no secuenciales, flujo de datos y el aspecto funcional.

Principio	Descripción	Aplicación a diagramas de flujo
Restricción	La notación debe restringir al usuario a formas comprensibles y correctas.	Al usar el conjunto restringido de bloques estructurados mejora la comprensión. Pero la notación no impide bloques no estructurados, que tienen que ser identificados examinando su configuración.
Redundancia	La información importante debe codificarse	Es muy difícil representar la misma información de distintas
Revelación y respuesta	La notación debe revelar el proceso que representa. Es preferible un ambiente que responda a la	Un ambiente interactivo, donde las imágenes se muevan en respuesta al usuario y se aprecie la dinámica del proceso, constituiría una herramienta muy importante para la resolución de
Revisabilidad (revisability)	Debe ser sencillo hacer modificaciones y correcciones.	La única manera sencilla de cambiarlos es borrando y volviendo a trazar los arcos. Esto puede desembocar en diagramas no estructurados y en flechas

Tabla 17. Principios de Fitter

Alcalde en [3] dice “La mayor utilidad que poseen las técnicas de diseño de programas (ordinogramas) es la de realizar el diseño con cierta independencia de las características particulares de los lenguajes de programación. Con ellos se consigue que un algoritmo de resolución de un problema pueda codificarse en cualquier lenguaje de cualquier máquina con las ventajas que supone la portabilidad del diseño.”

3.5.1.3. Desventajas y Problemática

Son muchas más las críticas y desventajas que se han vertido sobre los diagramas de flujo como herramienta de programación. El mismo Joyanes anota las siguientes desventajas:

2. Los diagramas de flujo se vuelven complejos y detallados por lo que suelen ser laboriosos en su planteamiento y dibujo.
3. Las acciones a seguir tras la salida de un símbolo de decisión, pueden ser difíciles de seguir si existen diferentes caminos.
4. No existen normas fijas para la elaboración de los diagramas de flujo que permitan incluir todos los detalles que el usuario desea introducir
5. Los diagramas de flujo son muy difíciles de seguir y de modificar

Otros autores como Perry en [43] señala “El dibujo de los diagramas de flujo requieren de mucho tiempo y papel. Aún cuando hay programas de diagramas de flujo que le pueden ayudar a dibujar y colocar símbolos, con frecuencia son limitados y no ofrecen la flexibilidad necesaria para la lógica de programación. Por lo tanto, es común recurrir al dibujo a mano de los diagramas. SI se termina un diagrama y se da cuenta que omitió dos símbolos críticos, tendrá que volver a dibujar una buena parte de éste. Debido a su naturaleza se invierte mucho tiempo en dibujar los diagramas de flujo, y algunas compañías no quieren que sus programadores hagan

esto cuando el pseudocódigo puede cumplir las mismas funciones y es más eficiente en cuestión de tiempo.”

Autores mexicanos como Levine en [66] expresa su crítica de la siguiente forma: “El problema de los diagramas de flujo - que no recomendamos- es que a medida que crece la complejidad (grado de anidamiento) de las proposiciones, también crece el detalle con que hay que dibujarlos. Esto llega a convertirlos en figuras fraccionadas (pues de otro modo no habría espacio suficiente en la hoja) difíciles de seguir y entender.”

Continúa Levine en [66] con lo siguiente: “Los diagramas de flujo podrían engañar de que una estructura está bien formada al representar una sola salida y una sola entrada pero esto puede ser solo una ilusión visual por la libertad en que se dibujan las líneas conectoras e incluso sugerir que se trata de instrucciones de tipo GO TO que son de tipo desestructurante y caótico en los programas e incluso terminar realmente no contar con una sola salida en el diagrama resultante.2

Recientemente se ha sostenido que el modelo de flujo de datos ofrece mayores ventajas que el de flujo de control, especialmente para programadores principiantes. Esta afirmación es bastante popular en los círculos de programación visual, a pesar de que las pruebas que la sostienen son escasas y contradictorias como lo señala Oberlander en [76]. La investigación experimental de Judith Good, en [46], ha mostrado que los novatos tienen un mejor desempeño general usando el modelo de flujo de control, aunque el de flujo de datos puede favorecer un análisis más abstracto del código.

La principal problemática que hay en el modelo imperativo de control de flujo es que no hay ahorro en los lenguajes de modelado y en consecuencia ya no se cumplen las expectativas de simplificación del código y productividad. Por ejemplo, en un diagrama de un sistema de Base de Datos, con un solo símbolo se pueden representar varias decenas de código, sin embargo, en un diagrama de flujo convencional un símbolo corresponde a una instrucción por lo que no hay verdadera aportación.

En conclusión, la problemática principal que hay en los lenguajes visuales programación imperativa son los siguientes:

Visibilidad.- Al crecer en complejidad el algoritmo crece el diagrama, el cual ya no puede apreciarse correctamente. Se comienza a hacer uso de conectores y eso afecta a navegabilidad conceptual de la lógica del algoritmo mismo.

Mantenibilidad.- Mantener actualizado el diagrama con respecto al código 3GL resultante requiere de esfuerzo y tiempo, y ambos son elementos que no se dispone en abundancia en al área de sistemas.

Complejidad.- Al crecer en elementos, un diagrama puede estar dibujado en forma tan confusa que oscurece su propósito y su lógica.

Ambigüedad.- El estándar ANSI actual para los diagramas de flujo utiliza igualmente el símbolo del rombo para representar condicionales y para representar ciclos abiertos del

tipo while y until lo cual aporta cierta dosis de ambigüedad en la simbología ocasionando problemas de interpretación visual.

Libertinaje estructural.- Los símbolos que representan el flujo de control (las flechas) no obedecen a regla alguna permitiendo llevar líneas de control a lugares totalmente inapropiados rompiendo la consistencia estructural y ocasionando confusión en su lectura e interpretación.

Legibilidad.- La simplicidad debe ser un objetivo prioritario. El utilizar símbolos para representar el flujo debe permitir que de un vistazo se pueda comprender cual es la secuencia, sin embargo, en la realidad sucede exactamente lo contrario; la complejidad, la ambigüedad y el libertinaje estructural ocasionan que los diagramas no puedan ser fácilmente leídos y comprendida su lógica.

Productividad.- Un símbolo corresponde a una instrucción por lo tanto no se mejora la eficiencia en la programación. Se requiere de igual o más esfuerzo hacer el diagrama que elaborar directamente el código 3GL.

3.5.1.4. Alternativas de Solución Propuestas

Ante el análisis de la problemática de la sección anterior se hacen en las siguientes propuestas:

1. Diseñar un lenguaje de modelado visual imperativo (LeMVI) inspirado en los símbolos actualmente conocidos y ampliamente difundidos en la cultura informática pero realizando algunas modificaciones para resolver problemas como el de la ambigüedad y falta de consistencia estructural.
2. Este lenguaje debe manejarse a dos capas: una abstracta que permita descripciones narrativas de interpretación propia para seres humanos y que puedan agrupar en un solo símbolo un conjunto amplio de instrucciones; y una concreta que permita la especificidad suficiente para soportar completo el teorema de Jacopini y el modelo de programación imperativa.
3. El lenguaje es simbólico pero también textual ya que al interior de cada símbolo se incluirán las especificaciones correspondientes.
4. Proponer una arquitectura que permita desarrollar un sistema informático como ambiente de programación que automatice el modelado de algoritmos utilizando el lenguaje propuesto (LeMVI), que considere las heurísticas que aporta este modelo (reglas de la lógica) y que siga la metodología de Microingeniería.

En la tabla 18 se exponen las estrategias de solución a los problemas ya citados combinando el lenguaje y el ambiente de programación junto con las heurísticas y metodología de Microingeniería de Software:

Problema	Definición	Solución
Visibilidad	No se puede ver el algoritmo completo por	Incluir un diagrama tipo "plano guía", el cual se visualiza en un panel adyacente al área de

Problema	Definición	Solución
	restricciones del espacio visual.	diseño y que tiene una estructura arborescente (inspirado en Warnier) al estilo de la estructura de directorio del explorador de Windows. Separación de secciones de código por implementación de módulos o de procedimientos. Éstos contarán con accesos directos. Inclusión de módulos abstractos que representan más de una instrucción. Los módulos abstractos pueden visualizarse en forma colapsable o en capas sobre puestas.
Mantenibilidad	Difícil y complicado de modificar.	Desarrollar la herramienta de modelado algorítmico implementada en un ambiente de programación y con capacidades bidireccionales a código 3GL.
Complejidad	Confuso en la interpretación por desorden en la representación gráfica.	Incluir un módulo de redibujo automático para crear limpieza y legibilidad.
Ambigüedad	Confuso en la interpretación por símbolos que pueden tener más de un significado	Elaborar un nuevo diseño sintáctico del lenguaje con correspondencia única entre símbolo y significado.
Libertad Estructural	Confuso en la interpretación ya que los flujos de control fácilmente pueden romper las estructuras formales definidas.	Considerar mecanismos de arrastrar y soltar con anclajes fijos en función de las heurísticas de este modelo denominadas "Reglas de la Lógica".
Legibilidad	La lógica de secuencia de ejecución no es clara por problemas de complejidad, ambigüedad y libertad estructural.	Lograr limpieza en el trazado por mecanismos de auto-redibujo. Agregar mecanismos de ejecución dinámica y depuración para permitir el seguimiento visual de ejecución y exploración de valores en memoria.
Productividad	Elaborar el diagrama requiere tanto tiempo y esfuerzo como el de escribir el código ya que hay una correspondencia de un símbolo a una instrucción.	Incluir en el diseño simbología para módulos abstractos que representan más de una instrucción representando ahorro en la codificación y simplicidad en la legibilidad. Sin embargo, si se aspira a poder ejecutar directamente el lenguaje estos módulos deberán ser especificados. Incluir procesos de ingeniería hacia adelante y hacia atrás con código 3GL en diversos lenguajes.

Tabla 18. Soluciones a problemas de modelado visual de algoritmos

3.5.2. Lenguaje Visual de Modelado

Alcalde en [3] explica que toda representación gráfica, de cualquier tipo que sea, debe cumplir las siguientes cualidades:

Sencillez.- Construcción de algoritmos de manera fácil y sencilla.

Claridad.- Fácil reconocimiento de todos sus elementos.

Normalización.- Debe tener normas para su construcción que sean completas y simples.

Flexibilidad.- Posibilidad de incluir posteriores modificaciones sin dificultades.

Combinando los modelos de Fitter, Alcalde y Kiper se obtiene un conjunto de especificaciones comparativas que el lenguaje de modelado debe cubrir, además de resolver las problemáticas identificadas en las secciones anteriores. La comparación entre sus propuestas se muestra en la 19.

Principios de diseño (Fitter)	Características (Alcalde)	Evaluación del lenguaje (Kiper)
Relevancia	Claridad	Facilidad de comprensión
-	Sencillez	Naturaleza visual
Restricción	Normalizado	Paradigma soportado
Redundancia	-	-
Revelación y respuesta	-	Escalabilidad
Revisabilidad	Flexible	Funcionalidad

Tabla 19. Especificaciones comparativas del lenguaje

Un lenguaje visual, de acuerdo a su definición, puede ser aplicado a un buen número de representaciones, en donde su significado está dado por la relación que guardan los elementos visuales, por ejemplo: las expresiones matemáticas, los diagramas de flujo, circuitos eléctricos, líneas de producción, etc.

Existen tres alternativas principales para la especificación de lenguajes visuales: la gramatical, la lógica y la algebraica.

La alternativa gramatical está basada en formalismos gramaticales utilizados en la especificación de lenguajes de cadenas. Dos formas representativas son: las gramáticas de grafos y las gramáticas de multiconjunto atribuidas. Una ventaja del formalismo de la gramática de grafos es que se entiende mejor debido a que se presenta mediante nodos y lados etiquetados, una desventaja de este formalismo es que requiere de la fase de análisis inicial “léxica” para reconocer las relaciones entre los símbolos terminales que son importantes y que deben ser conservados como lados en el grafo inicial. Las gramáticas de multiconjunto atribuidas dificultan la manera en que se definen los elementos que implementa la gramática de grafos, siendo una desventaja de estas gramáticas, que son demasiado expresivas puesto que ellas son computacionalmente adecuadas.

La alternativa lógica utiliza lógica matemática de primer orden u otras formas de lógica matemática que frecuentemente provienen de inteligencia artificial. Está basada en alternativas que son usualmente lógicas espaciales, que axiomatizan las diferentes relaciones posibles entre objetos. Una de las ventajas que proporciona esta opción es que el mismo formalismo puede ser usado para especificar la sintaxis y semántica de un diagrama.

Otra opción para especificar lenguajes visuales es usar la especificación algebraica. Ésta consiste en una composición de funciones para la construcción de imágenes complejas a partir de

los elementos más simples de la imagen. La idea principal de esta opción consiste en mapear el dominio definido, utilizando estructuras de tipo de datos abstractos y definiendo los tipos de funciones y operaciones de predicados con estas estructuras, para así especificar las operaciones en el dominio de aplicación. El concepto de jerarquía en los tipos es el principal tópico en la especificación algebraica, ya que pueden ser considerados los tipos de manera análoga a los objetos, cuando los tipos son un sustituto de las clases.

La especificación de un lenguaje visual está dada de la siguiente manera: primero debe delimitarse el dominio de aplicación, después identificar los elementos que forman parte de este dominio, definir las relaciones válidas dentro del mismo y finalmente determinar el comportamiento de las relaciones.

Para el caso de este lenguaje de modelado, LeMVI, se utilizará un modelo gramatical dividido en dos partes: una visual y una textual.

3.5.2.1. Gramática Visual

Según Aho en [2], una gramática independiente del contexto está formada por cuatro componentes:

$$G = (N, T, S, P)$$

Donde

- N Es el conjunto de símbolos no terminales
- T Es un conjunto de componentes léxicos denominados símbolos terminales.
- S Es un símbolo no terminal utilizado como inicial
- P Es un conjunto de reglas de producción en el que cada producción consta de un no terminal, llamado lado izquierdo de la producción, un símbolo de correspondencia (en este caso se utilizará ::=) y una secuencia de componentes léxicos terminales, no terminales o ambos, llamado lado derecho de la producción.

El lenguaje de modelado propuesto tiene dos capas. La primera corresponde a un nivel abstracto y de naturaleza conceptual; la segunda corresponde a un nivel concreto y de naturaleza imperativa. Ambas capas se mezclan en un solo diagrama aportando flexibilidad expresiva. En la figura 64 se muestran los conjuntos N, T y S de la gramática de ambas capas.

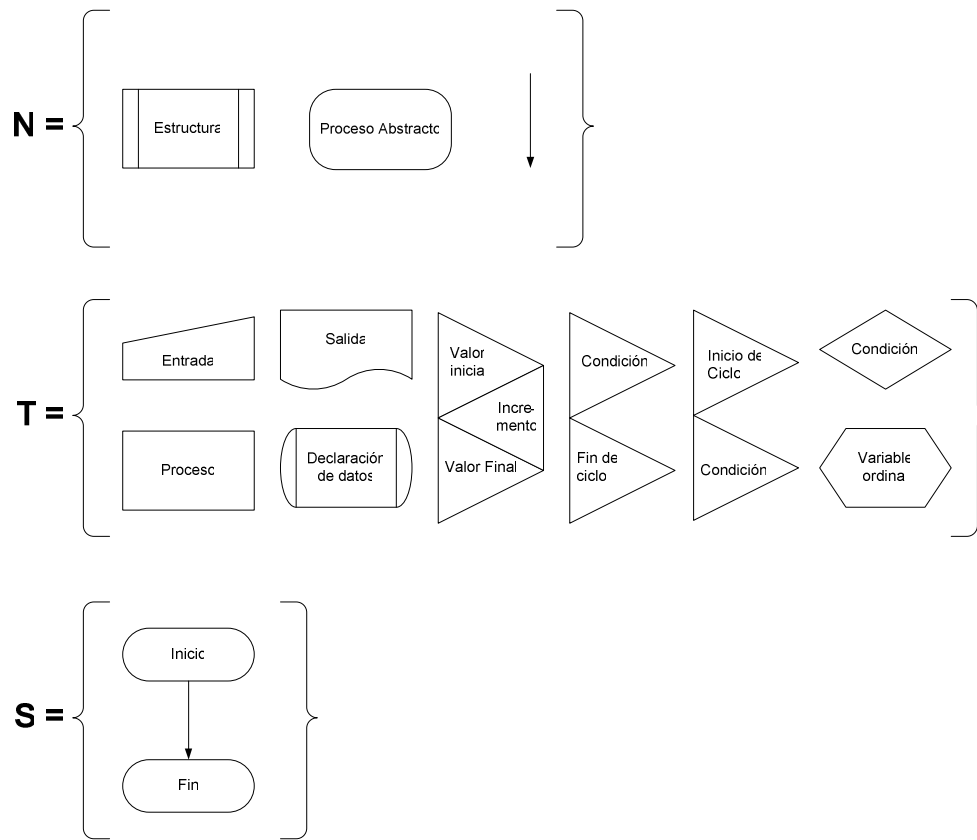


Figura 64. Gramática de LeMVI

La capa abstracta puede concretarse para adaptarse al modelo imperativo y volverse funcional pero puede conservarse como agrupador de varias estructuras.

Adicional al conjunto gramatical del lenguaje se agrega, como parte de la capa abstracta no ejecutable un símbolo para representar comentarios, y notas que sirva como mecanismos de documentación. Se decidió utilizar la misma sintaxis de UML para este propósito por lo que la figura 65 muestra la simbología correspondiente.

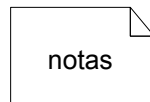


Figura 65. Elemento documental de LeMVI

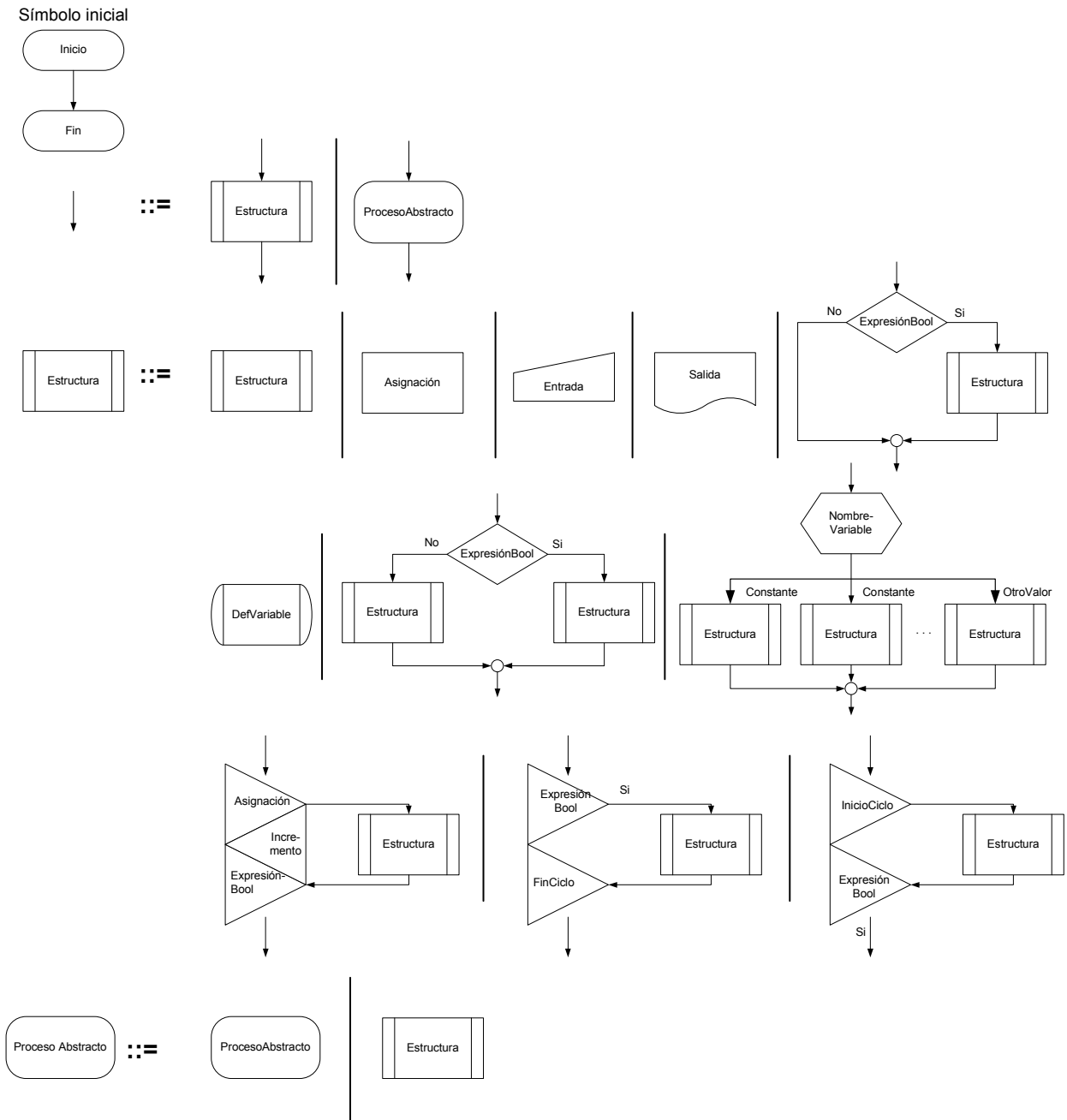


Figura 66. Reglas de producción de la gramática de LeMVI (conjunto P)

En la figura 66 se muestra el conjunto P, reglas de producción, dónde se explica la integración de los léxicos involucrados. Como puede observarse, LeMVI maneja estructuras completas como símbolos terminales en los cuales se define de antemano al flujo de control a través de la colocación de las flechas de seguimiento de flujo. Las flechas de flujo de control no son editables por el usuario ni se pueden ubicar discrecionalmente. Su colocación es fija. Puede observarse en las estructuras condicionales que hay un símbolo auxiliar (un pequeño círculo conector dónde convergen las flechas). Este se utiliza para conservar la integridad de la estructura. Ya que una flecha puede ser convertida en una estructura o en un procedimiento abstracto, este pequeño

conector de convergencia determina el lugar dónde la figura debe dibujarse y no otro discrecional.

3.5.2.2. Gramática de Enunciados

Dada la naturaleza combinada del lenguaje de modelado propuesto (LeMVI), cada figura gráfica incluye necesariamente una expresión textual. En la figura 66 se muestran las reglas de producción y éstas contienen los símbolos terminales. Cada figura incluye concepto textual que debe ser definido.

Palabras reservadas por LeMVI son: AND, OR, NOT, DIV, MOD, integer, real, character, boolean, Trae, False, InicioCiclo, FinCiclo. No se distingue entre mayúsculas y minúsculas.

La gramática de enunciados para los símbolos terminales es la siguiente:

```

Asignación ::= Variable = [Expresión | ExpresiónBool] (a) ;
Entrada ::= ExpresiónEntrada (, . ExpresiónEntrada)*
ExpresiónEntrada ::= Cadena, NombreVariable (b)
Salida ::= ExpresiónSalida (, . ExpresiónSalida)*
ExpresiónSalida ::= (Cadena) [Expresión | ExpresiónBool |
                          NombreVariable]
Cadena ::= "(Letra | Dígito | Caracter)*"
DefVariable ::= TipoVariable NombreVariable(, NombreVariable)*
TipoVariable ::= integer | real | character | boolean
Expresión ::= Termino (OperadorAritmético Termino)*
Termino ::= [NombreVariable | ConstanteNumérica]
OperadorAritmético ::= [+ | - | * | / | ** | DIV | MOD] (d) (e)
ConstanteNumerica ::= Número
Número ::= (-) Dígito* ((.) Dígito*)
Dígito ::= [0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]
Letra ::= [a | b | c | ... | Z]
Carácter ::= [; | # | $ | ... | ¿]
ConstanteBool ::= [True | False]
NombreVariable ::= Letra (Letra | número | _)*
Incremento ::= [Número | NombreVariable]
ExpresiónBool ::= (NOT) '('('TérminoBool')' ( OperadorLógico (NOT)
                          '('('TérminoBool')' )*) (c)
OperadorLógico ::= [AND | OR]
OperadorRelacional ::= [= | < | > | >= | <= | <>]
TérminoBool ::= [ConstanteBool | Condición]
Condición ::= NombreVariable OperadorRelacional [NombreVariable |
                          ConstanteNumérica | Cadena]

```

En esta gramática cabe resaltar lo siguiente:

(a) Aunque los operadores de igualdad y asignación se representan igual, la “sutileza sintáctica” se reduce debido al símbolo que la contiene.

(b) Las cadenas se pueden utilizar únicamente para hacer más presentable los procesos de entrada y salida.

(c) A pesar de que NOT es un operador unario, la negación tiene una baja prioridad puesto que únicamente se permite aplicarla a valores booleanos.

- (d) La división entera tiene su propio operador, similar al de módulo.
- (e) Representar la potencia como función es poco consistente. Usar “^” conlleva el riesgo de ser más difícil de localizar en el teclado.

Aunque no es evidente en la gramática, todos los operadores funcionan únicamente con operandos del mismo tipo. La tabla 20 muestra los tipos de datos que tienen las operaciones como operandos y resultado.

Operador	Operandos	Resultado
+ - * **	Números	Real (1 o 2 operandos reales) Entero (2 operandos enteros)
/	Números	Real
DIV MOD	Números	Entero
< > <= >=	Números	Booleano
= <>	Cualquiera	Booleano
OR AND NOT	Booleanos	Booleano

Tabla 20. Operadores y operandos

El lenguaje LeMVI tiene tipos fuertes (es strongly typed). Esto implica que no se permite hacer operaciones entre tipos distintos (con la excepción de enteros con reales). A diferencia de lenguajes como C, los valores alfanuméricos o lógicos no tienen una representación numérica. Cualquiera

Los nombres de variables no pueden repetirse y tienen que ser distintos de las palabras reservadas.

Las variables tienen una tercera propiedad y es que su valor solamente es accesible en tiempo de ejecución. Es inválido hacer referencia a una variable antes de asignarle un valor (aplicando la Regla de Existencia de las heurísticas)

3.5.2.3. Semántica

A continuación se presentan los modelos semánticos del conjunto de símbolos terminales y no terminales. Se describirá la semántica a través de narrativa simple.

Símbolo Inicial.- representado por dos nodos interconectados por una flecha de control de flujo. El primer nodo representa el inicio del algoritmo y el último su final.

Flecha de flujo de control.- Representa la secuencia de ejecución del lenguaje. Dónde sea que haya una flecha se podrá colocar una estructura o un proceso abstracto. Esto está basado en la regla seis (Creación) del conjunto de heurísticas denominado Reglas de la Lógica.

Estructura.- elemento No terminal que puede ser sustituido por una secuencia de estructuras o de procesos abstractos. Las estructuras validas son los símbolos denominados Acciones (entrada, proceso y salida) o por las estructuras condicionales y cíclicas.

Asignación.- En ella se realizan los cálculos aritméticos y la asignación de resultados a la variable dependiente.

Entrada.- Realiza la lectura de valores desde el teclado. Se acompaña de una cadena para colocar en ella una pregunta aclaratoria sobre el valor que se solicita al usuario.

Salida.- Realiza la emisión de resultados al monitor. Los resultados pueden ser acompañados de cadenas explicativas referentes al valor mostrado. Los valores pueden ser numéricos, alfabéticos o lógicos.

Definición de Variables.- Reserva el espacio en la memoria para almacenar valores de alguno de los tipos soportados.

Condición simple.- Se evalúa la expresión booleana, en caso de que su resultado sea verdadero (True) se ejecuta la estructura señalada en flujo de control con la palabra “si”, al terminar este bloque se continúa hacia el punto de convergencia. En caso que la expresión booleana sea evaluada como falsa (False) se continúa el flujo por el señalamiento con la palabra “no” y de avanza directamente al punto de convergencia.

Condición doble.- Se evalúa la expresión booleana, en caso de ser verdadera (True) se continúa por el flujo señalado con la palabra “si”, se ejecuta la estructura señalada en ese flujo y se continúa al punto de convergencia. En caso de que la expresión booleana sea evaluada como falsa (False) se continúa el flujo por el señalado con la palabra “no”, se ejecuta el bloque de de instrucciones de la estructura ahí definido y se continúa al punto de convergencia.

Multicondicional.- Se toma el valor de la variable y se compara contra el primer valor constante de la ruta de flujo de control del extremo izquierdo, si son iguales se ejecuta ese bloque; si no son iguales se compara la variable con el valor constante de la segunda ruta alternativa, si son iguales se ejecuta la segunda ruta; si no son iguales se sigue comparando con las siguientes rutas siempre leyéndose de izquierda a derecha. Si se terminan las rutas se ejecuta la señalada con la palabra “Otro valor”.

Ciclo cerrado.- Se asigna a una variable de control un valor inicial y se continúa el flujo hacia la derecha para ejecutar el bloque indicado en la estructura. Al terminar la ejecución se regresa al ciclo cerrado para ejecutar el incremento (o decremento según el signo) de la variable de control. Una vez actualizado su valor se realiza la evaluación de la expresión booleana. Si el resultado es verdadero (True) se toma la ruta de la derecha y se repite la ejecución del bloque definido en la estructura anidada. Si el resultado es falso (False) se continúa hacia abajo con la siguiente sentencia que corresponda.

Ciclo abierto con condición al inicio.- Se evalúa la expresión booleana, si el resultado es verdadero (True) se continúa hacia la derecha, por el flujo señalado con la palabra “si”, y se ejecuta el bloque anidado y definido por la estructura. Al terminar se regresa al ciclo y se repite la evaluación de la expresión booleana. Cuando la expresión es evaluada como falsa se continúa el flujo hacia abajo dando por terminado el ciclo y continuando con la siguiente estructura.

Ciclo abierto con condición al final.- Al iniciar el ciclo inmediatamente se toma la ruta de la derecha para ejecutar el bloque anidado y definido por la estructura. Se regresa al ciclo y se evalúa la expresión booleana, si la expresión es verdadera (True) se termina el ciclo y se continua hacia abajo con la estructura siguiente. Si la expresión booleana es evaluada como falsa (False) se repite la ejecución de la estructura anidada para posteriormente volver al ciclo y evaluar de nuevo la expresión booleana.

3.5.3. Arquitectura del Sistema

Una vez que se tiene la metodología de microingeniería, las heurísticas de diseño de algoritmos y la definición del lenguaje de modelado se propone el diseño de una herramienta computacional que pueda asistir a las personas en el diseño de algoritmos. Esta arquitectura se le ha denominado DAAC por sus siglas de Diseño a Algoritmos Asistido por Computadora. Esta arquitectura también soporta el modelo de habilidades que se analiza en la sección 3.6.

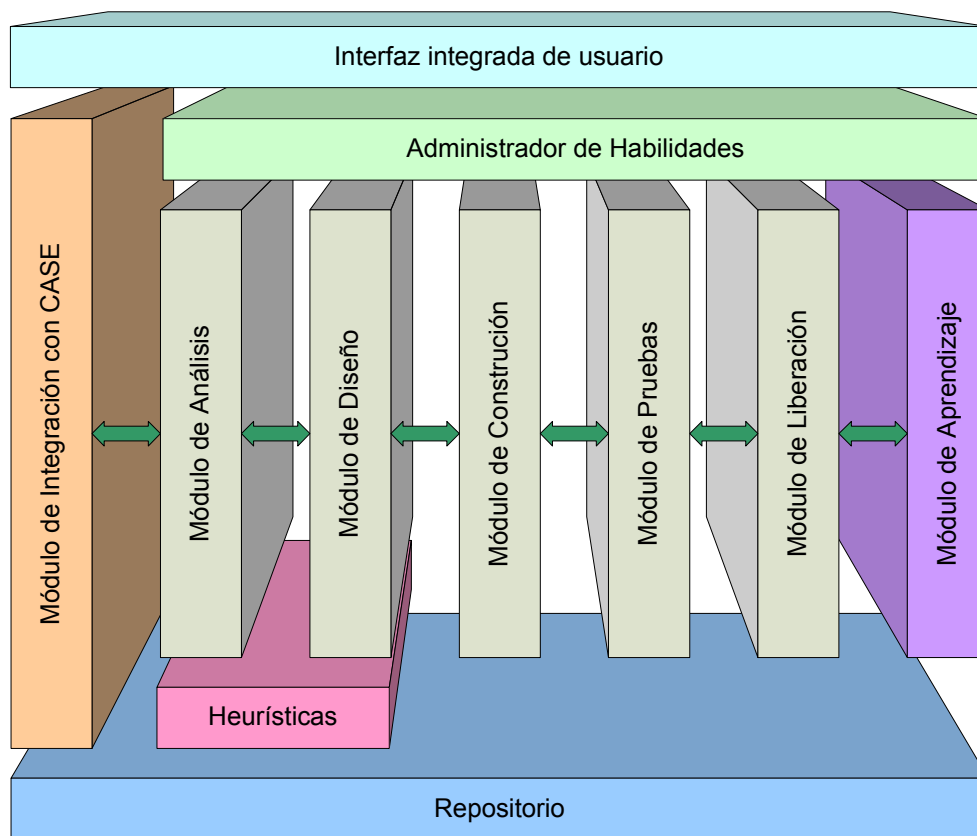


Figura 67. Arquitectura conceptual

3.5.3.1. Arquitectura DAAC

La arquitectura conceptual incluye los elementos del modelo de construcción de algoritmos apoyado en heurísticas. Se encuentra un componente para modelar las heurísticas, otro para las habilidades y uno más para la metodología de microingeniería que contempla cada una de las etapas del ciclo de vida. Además, se incluye un módulo de interfaz con sistemas del tipo CASE y

uno de administración del aprendizaje dependiendo de escenario de operación del sistema. Finalmente se considera una interfaz integrada y un repositorio de datos.

La figura 67 muestra la arquitectura modular conceptual en dónde se pueden apreciar los módulos y las relaciones entre ellos.

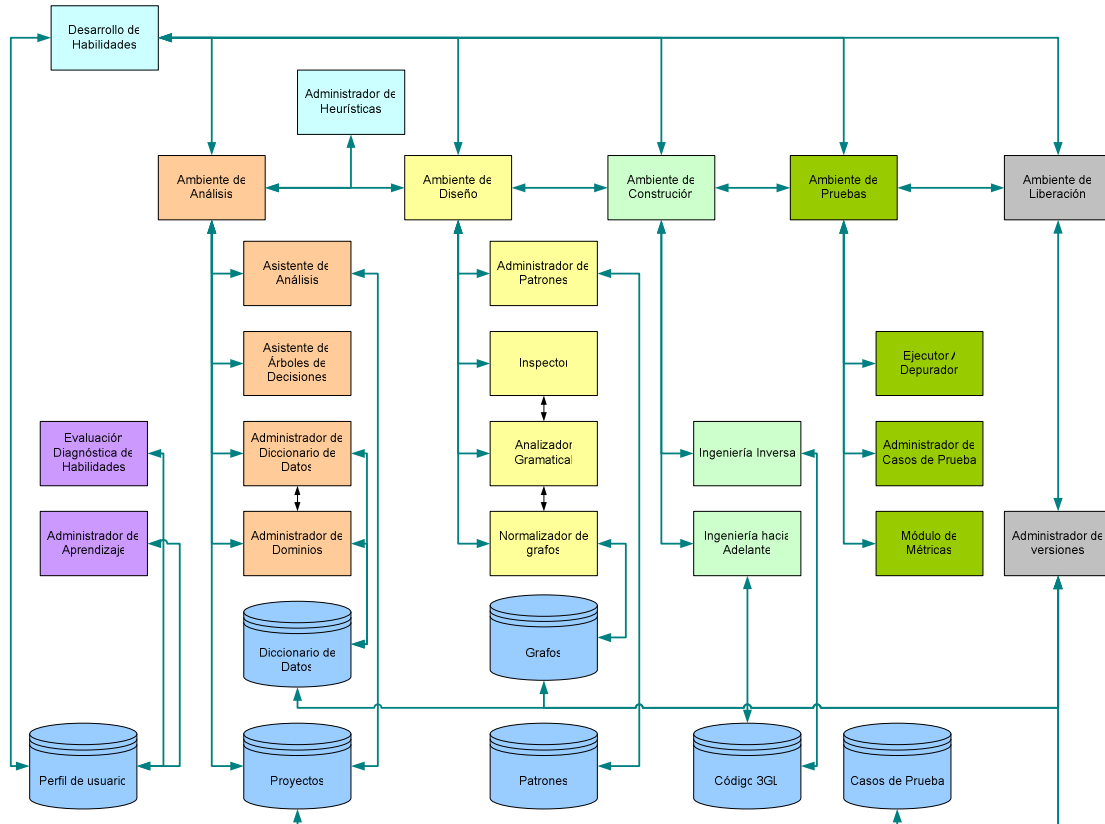


Figura 68. Arquitectura DAAC

La figura 68 detalla, para cada uno de los módulos conceptuales, sus módulos funcionales y las interfaces que hay entre ellos.

3.6 Dimensión Humana: Modelo de Habilidades

El enfoque con que este trabajo se ha abordado es holístico, y dentro de las dimensiones que se han identificado está el modelo teórico (explicado en la sección 3.4), el herramental (explicado en la sección 3.5) y finalmente el de las habilidades. En esta sección se aborda un modelo de la inteligencia, se identifican y clasifican las habilidades requeridas para la programación bajo este modelo, y se hace un análisis sobre los enfoques educativos para poder desarrollar dichas habilidades.

3.6.1. La Lógica de Programación como una Habilidad Mental

La Lógica de Programación es una habilidad mental superior ya que se requiere para su correcto desarrollo de otras habilidades mentales como la capacidad de abstracción, análisis, síntesis, inferencia, diseño, etc.; todo esto acompañado de fuertes dosis de creatividad e ingenio. Sin embargo, y a pesar de su complejidad, es posible desarrollarla a través de reglas, estructuras y práctica.

Habilidad viene del término latino *habilis* que significa manejable [48], el diccionario define el término como la capacidad, disposición, gracia o destreza para hacer algo; en este caso programar. A su vez programar consiste en dividir cualquier tarea compleja en una sucesión de ordenes simples que estén al alcance de la computadora [80]. Finalmente, Lógica proviene del término griego *logos* (λόγος) que significa tratado o conocimiento [48]. El diccionario la define cómo la ciencia que estudia las leyes y modos del conocimiento científico mediante el raciocinio. La Lógica Formal se entiende como la ciencia que estudia los procesos del pensamiento con referencia a la validez de sus conclusiones sacadas de ciertos tipos de premisas las cuales pueden ser verdaderas o falsas. La forma en que se combinan las premisas determinan el tipo de silogismo y existen reglas claras para su construcción.

Partiendo de la información expuesta se propone considerar a la Lógica de Programación como una habilidad mental utilizada para implementar soluciones computacionales (programas) orientados a ejecutar tareas complejas funcionalmente verdaderas (que hagan lo que se espera de ellos - Conclusiones válidas) a través de la combinación de diferentes estructuras de control y estructuras de datos (premisas) combinadas utilizando una serie de reglas formales (heurísticas o “Reglas de la Lógica”).

Las habilidades, a diferencia de los conocimientos, se adquieren con la experiencia vivencial de su praxis. Este es el principal problema de varios enfoques educativos ya que pretenden que la persona “aprenda” las definiciones y conceptos de los elementos que forman la Lógica de Programación y los observen en la ejecución de unos cuantos ejercicios. Esta propuesta contempla que, además de realizar las descripciones teórico-conceptuales, se acompañe de un modelo de desarrollo de habilidades, apoyados en ejercicios, que demuestre tanto la operación de las estructuras de control como de las estructuras de datos básicas a través de la aplicación de las ya comentadas Reglas de la Lógica y otras herramientas históricamente validadas. En pocas palabras consiste en privilegiar el aprendizaje basado en la experimentación y vivencia.

La Lógica de Programación es una habilidad mental superior y se desarrolla a través de experiencias (prácticas) que deben ser planeadas en temática y complejidad para que estén vinculadas con su realidad y el resto de sus materias de estudio.

3.6.2. Un Modelo para Clasificar las Habilidades

En la sección 2.3.3.10 se hizo una introducción al modelo de la inteligencia de Guilford [11]. Este modelo permite combinar la información y sus elementos con los procesos mentales que operan sobre ellos y que forman las habilidades de la inteligencia. En la figura 69 se muestra dicho modelo.

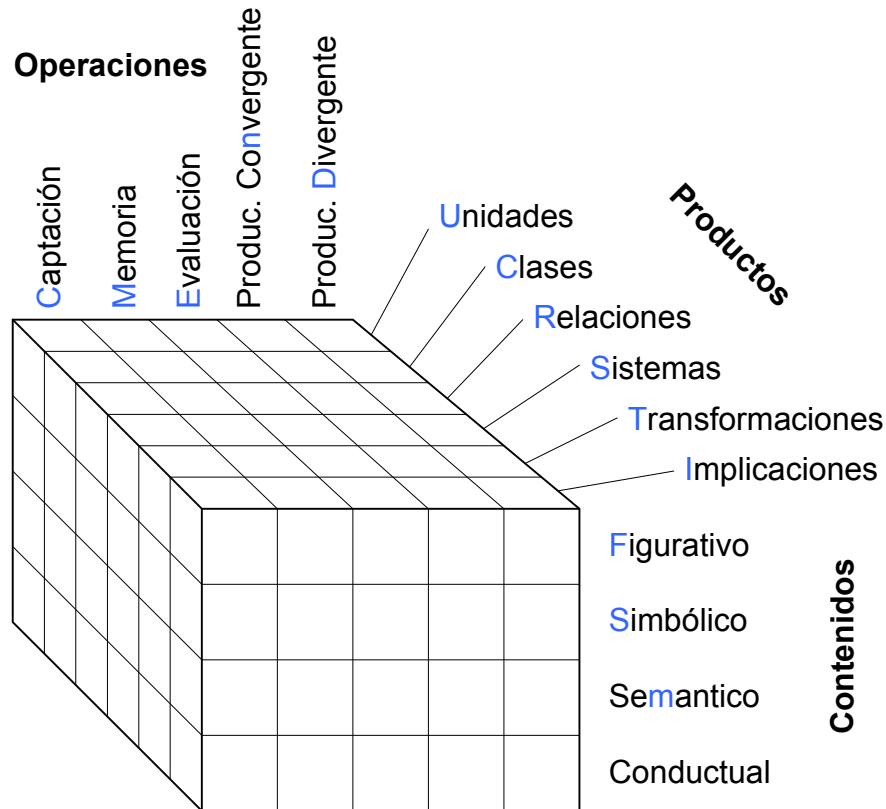


Figura 69. Estructura de la inteligencia de Guilford

Como se puede observar, el modelo está dividido en tres ejes: operaciones, productos y contenidos; y cada uno tiene sus propios elementos. El cruce de cada componente de cada eje identifica las habilidades. En cada eje, el nivel de complejidad se incrementa progresivamente. Por ejemplo, en el eje de operaciones mentales primero se capta, luego se memoriza, luego se evalúa, etc. cada etapa de cada eje está fincada en que la previa ya ha sido desarrollada. Por lo tanto no es posible hacer una producción convergente si primero no se ha hecho una evaluación. Esto sucede en todos los ejes y proporciona un mecanismo incremental de desarrollo de las habilidades.

Las habilidades, que se denominarán intelectuales, están formadas por un componente de cada eje poniendo primero la operación, seguida del producto y finalmente el contenido. Por ejemplo la primera habilidad es la captación de unidades figurativas (denominada CUF), la segunda es la captación de clases figurativas (CCF), la tercera es la captación de relaciones figurativas (CRF), etc. Como puede verse, primero se incrementa el eje de productos, luego se incrementa el eje de los contenidos y finalmente se incrementa el eje de las operaciones.

A continuación se presenta una descripción de cada elemento de cada eje de la inteligencia según Guilford.

3.6.2.1. Contenidos de la Información

Toda información tiene necesariamente un contenido específico que diferenciará la estructura intrínseca de los datos que posteriormente procesará la inteligencia. Hay cuatro tipos de contenidos:

1. Información figurativa.- Es la información que se capta directamente en forma concreta y tangible, sin necesidad de interpretación o decodificación; tiene puntos de partida sensoriales. Por ejemplo, al ver un semáforo, la información figurativa nos indica únicamente tres luces de colores, sin ninguna interpretación o significado.
2. Información simbólica.- Los símbolos exigen una traducción o decodificación, pues llevan a otra realidad, diferente a la que se percibe directamente. Normalmente sintetizan otros datos o informaciones que llevarían más tiempo o espacio. Esta síntesis obliga a abstraer en forma considerable, por lo que exige que el ser humano haya evolucionado para poder captarla. En el caso de observar un semáforo, cada color tiene un símbolo específico. Muchos mensajes de tránsito carretero y ciudadano requieren del manejo adecuado de la simbología.
3. Información semántica.- Se refiere al significado que se percibe en una información. Por ejemplo el significado del rojo en el semáforo significa detenerse, el verde avanzar, etc. Guilford sugiere algunos detalles para afinar la comprensión de ese concepto. La captación de elementos esenciales de una información proporciona el significado denotativo⁸; a la contribución de características que enriquecen una información se conoce como significado connotativo⁹.
4. Información Conductual.- Se refiere a la manifestación de las actitudes, relaciones interpersonales y los sentimientos que interactúan. Parte de la información figurativa se traduce directamente en información conductual tales como las expresiones faciales que manifiestan sentimientos y es lo que se denomina “lenguaje corporal”.

Isauro Blanco, en [11] explica que Mary Meeker no emplea la categoría Conductual por estar incluida implícitamente en el contenido simbólico y, sobre todo semántico por lo que propone un modelo simplificado en el cual únicamente se consideran los tres primeros tipos de contenido. Este trabajo clasificará las habilidades intelectuales utilizando la versión simplificada de Mary Meeker.

3.6.2.2. Productos de la Información

Los productos se refieren a la estructura de la información: son formas de construcción mental. Hay seis diferentes tipos de productos, que tienen una graduación en la complejidad del aprendizaje y el manejo:

⁸ Cada palabra cuenta con un significado-base que viene a ser la acepción primera con la que cada palabra es definida en el diccionario. Se trata del sentido más común y generalizado de cada palabra. Ese significado primario se denomina denotación, conjunto de semas unidos de forma constante y estable a cada unidad léxica.

⁹ Las palabras pueden adoptar diferentes significados dependiendo del contexto en que se utilice. El fundamento de la connotación está vinculado a la polisemia del lenguaje.

1. Unidades: Es el producto básico de la información. Es muy concreto y no requiere de abstracción o interpretación, la captación de unidades depende de los sentidos.
2. Clases: Es la agrupación de unidades en una categoría, por tener uno o mas atributos en común.
3. Relaciones: Son las diferentes conexiones que se elaboran entre unidades pertenecientes a diferentes clases; tales relaciones pueden ser de diferentes tipos y exigen mayor capacidad de abstracción
4. Sistemas: La relación de relaciones forma un sistema, donde todas las unidades funcionan como un todo integrado. Es un producto más complejo que los anteriores. Generalmente, un sistema implica un orden o secuencia de unidades. El lenguaje es un sistema semántico; la matemática es un sistema simbólico. La programación combina ambos.
5. Transformaciones: Son los cambios que mejoran un sistema y que implica un dominio de éste para efectuar las modificaciones que generarán a su vez, un nuevo elemento.
6. Implicaciones: Se refieren a la previsión de las consecuencias y secuelas que se derivan de las trasformaciones elaboradas. Este producto se orienta hacia el futuro de las acciones generadas sobre la información, fundamentadas en la lógica.

Los productos de información son graduales: las unidades son el producto elemental, a partir del cual se estructura toda la información; las clases requieren de mayor abstracción y son más complejas que las unidades, hasta llegar al producto más complicado que es la implicación y que se refiere al futuro, que aún no existe, pero que se construirá con las acciones del presente.

Una vez analizada la información en sus dos modalidades: contenidos y productos; se complementa el modelo intelectual con los procesos, que se refieren a la intervención de la inteligencia sobre los contenidos y productos de la información.

3.6.2.3. Procesos Intelectuales

En la figura 70 se muestra el diagrama de los procesos intelectuales que realiza la mente sobre la información.

1. Captación de la información.- Este paso se refiere a la forma en que se tiene el primer contacto con la información que va a ser procesada; generalmente la captación depende de la entrada sensorial.
2. Memoria de la información.- Después de que se capta la información, esta fluye a un banco de datos que se llama memoria. Guilford explica que su capacidad de “almacenamiento” es prácticamente ilimitada pero no así la posibilidad de evocación. La capacidad de recordar algo depende de la clasificación que se haga durante la captación y de las asociaciones o relaciones que se realicen.
3. La evaluación de la información.- Cuando la realidad plantea retos, dificultades, problemas o interrogantes, se requiere que la inteligencia evalúe la información que tiene acumulada y seleccionar los mejores datos disponibles para dar respuesta a las preguntas que se le presentan.

4. Producción convergente (solución de problemas).- La captación, la memoria y la evaluación son procesos que se realizan al interior de la mente y presentan su efecto directo en la solución de problemas.

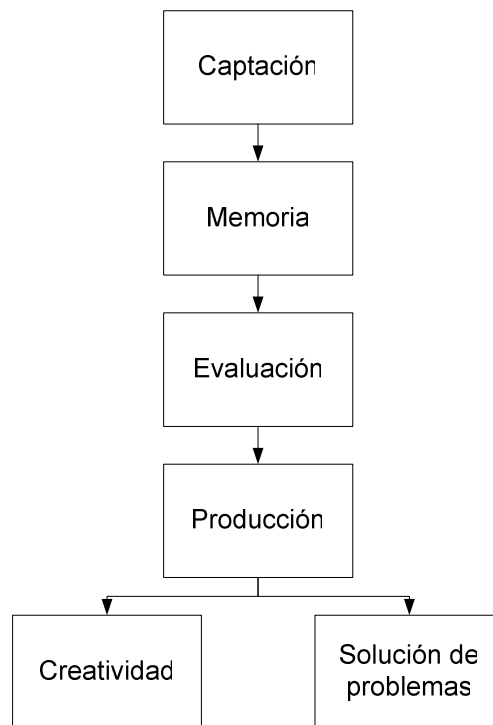


Figura 70. Flujo de procesos mentales

5. Producción divergente (creatividad).- Otra de las formas de producción es la creatividad para encontrar nuevas alternativas de solución a problemas desconocidos.

El desarrollo de la inteligencia se incrementa en el reto, la dificultad y la ambigüedad. La claridad no propicia pensamiento; la facilidad castra a la mente. Lógicamente no se propone la confusión ni la improvisación, sino de una metodología que gradualmente estimula a la inteligencia como el ejercicio físico y el gimnasio desarrolla la musculatura.

3.6.2.4. Habilidades Intelectuales en la Programación

En la estructura de la inteligencia, Guilford plantea que las habilidades intelectuales condicionan las diferentes funciones del ser humano. Este enfoque propone soluciones centradas más en las causas que en los efectos. Por ejemplo, si una persona presenta dificultad en matemáticas (la función) se debe fortalecer la estimulación de las habilidades intelectuales que condicionan el aprendizaje de las matemáticas. Gran parte del problema en el aprendizaje radica en atender masivamente los efectos de las deficiencias en lugar de corregir las causas.

Ya se explicó que cada habilidad está formada por un componente de cada eje. Si en el modelo original son cinco operaciones, seis productos y tres contenidos; se tienen un total de 90 habilidades intelectuales fundamentales, sin embargo, como algunas se subdividen en aspectos connotativos y denotativos llegan a más de 100. El análisis detallado de cada uno se puede

consultar en [11]. Únicamente se mencionarán las definiciones de las habilidades que están implicadas con el uso del lenguaje, de las matemáticas y de la lógica que están directamente relacionadas con la programación.

Las habilidades intelectuales derivadas de la captación de la información se presentan en orden de desarrollo en la tabla 21.

Clave	Nombre	Descripción
CUS	Captación de Unidades Simbólicas	Reconocimiento Simbólico
CCS	Captación de Clases Simbólicas	Conceptos numéricos
CRS	Captación de Relaciones Simbólicas	Relaciones numéricas
CSS	Captación de Sistemas Simbólicos	Sistemas numéricos
CTS	Captación de Transformaciones Simbólicas	Transformaciones simbólicas
CIS	Captación de Implicaciones Simbólicas	Implicaciones simbólicas
CUM	Captación de Unidades Semánticas	Vocabulario
CCM	Captación de Clases Semánticas	Conceptualización
CRM	Captación de Relaciones Semánticas	Razonamiento analógico
CSM	Captación de Sistemas Semánticas	Comprensión verbal
CIM	Captación de Implicaciones Semánticas	Implicaciones verbales

Tabla 21. Habilidades intelectuales de captación de información

Las habilidades intelectuales derivadas de la memoria de la información se muestran en orden en la tabla 22.

Clave	Nombre	Descripción
MUF	Memoria de Unidades Figurativas	Evocación detalles figurativos
MTF	Memoria de Transformaciones Figurativas	Evocación de transformaciones figurativas
MUS	Memoria de Unidades Simbólicas	Evocación detalles simbólicos
MCS	Memoria de Clases Simbólicas	Conceptos simbólicos
MRS	Memoria de Relaciones Simbólicas	Relaciones simbólicas
MSS	Memoria de Sistemas Simbólicos	Secuencias simbólicas
MIS	Memoria de Implicaciones Simbólicas	Implicaciones simbólicas
MRM	Memoria de Relaciones Semánticas	Relaciones semánticas
MSM	Memoria de Sistemas Semánticos	Secuencias verbales

Tabla 22. Habilidades intelectuales de la memoria

Las habilidades intelectuales de la evaluación de la información se presentan, también en orden de estimulación en la tabla 23.

Clave	Nombre	Descripción
ECS	Evaluación de Clases Simbólicas	Discriminación conceptualización de símbolos
ERS	Evaluación de Relaciones Simbólicas	Análisis relaciones simbólicas
ESS	Evaluación de Sistemas Simbólicos	Evaluación sistemas simbólicos
ETS	Evaluación de Transformaciones Simbólicas	Discriminación de transformaciones simbólicas
EIS	Evaluación de Implicaciones Simbólicas	Análisis de implicaciones simbólicas

Clave	Nombre	Descripción
ECM	Evaluación de Clases Semánticas	Discriminación conceptual
ERM	Evaluación de Relaciones Semánticas	Evaluación de conceptos que se relacionan
ESM	Evaluación de Sistemas Semánticos	Evaluación de secuencias semánticas
ETM	Evaluación de Transformaciones Semánticas	Transformaciones semánticas
EIM	Evaluación de Implicaciones Semánticas	Análisis de implicaciones semánticas

Tabla 23. Habilidades intelectuales de la evaluación de información

Sigue el turno de las habilidades involucradas con la programación con la operación de producción convergente mostrada en la tabla 24.

Clave	Nombre	Descripción
NCS	Producción Convergente de Clases Simbólicas	Clasificación simbólica
NRS	Producción Convergente de Relaciones Simbólicas	Aplicación relaciones simbólicas
NSS	Producción Convergente de Sistemas Simbólicos	Aplicación sistemas simbólicos
NTS	Producción Convergente de Transformaciones Simbólicas	Transformaciones simbólicas
NIS	Producción Convergente de Implicaciones Simbólicas	Solución problemas simbólicos
NRM	Producción Convergente de Relaciones Semánticas	Aplicación razonamiento analógico
NSM	Producción Convergente de Sistemas Semánticos	Reproducción secuencias semánticas
NTM	Producción Convergente de Transformaciones Semánticas	Transformaciones verbales
NIM	Producción Convergente de Implicaciones Semánticas	Implicaciones semántica

Tabla 24. Habilidades de producción convergente para la programación

Las operaciones de producción divergente (creatividad) que están asociadas con la programación se muestran en la tabla 25.

Clave	Nombre	Descripción
DUS	Producción Divergente de Unidades Simbólicas	Fluidez simbólica
DCS	Producción Divergente de Clases Simbólicas	Conceptualización simbólica
DRS	Producción Divergente de Relaciones Simbólicas	Relaciones simbólicas
DSS	Producción Divergente de Sistemas Simbólicos	Sistematización simbólica
DTS	Producción Divergente de Transformaciones Simbólicas	Transformaciones simbólicas
DIS	Producción Divergente de Implicaciones Simbólicas	Inferencias simbólicas

Clave	Nombre	Descripción
DSM	Producción Divergente de Sistemas Semánticos	Sistematización semántica
DTM	Producción Divergente de Transformaciones Semánticas	Transformaciones verbales
DIM	Producción Divergente de Implicaciones Semánticas	Inferencias semánticas

Tabla 25. Habilidades de producción divergente para la programación

Cada una de las habilidades intelectuales citadas en las tablas anteriores condicionan la competencia de la programación por lo deben ser desarrolladas y fortalecidas como se explica en [11]. Se han identificado 49 totales pero si se considera la ley de pareto las de mayor impacto son 9. Se explican con detalle en la sección 3.6.4.

3.6.3. Modelo de Desarrollo de Habilidades

Los modelos de desarrollo de habilidades están fuertemente vinculados con los procesos educativos que se han desarrollado durante el tiempo. Actualmente, uno de los modelos más discutido y con mayor auge en el desarrollo curricular a nivel mundial es el modelo enfocado a las Competencias.

La noción de competencia profesional pretende mejorar la relación del sistema educativo con el productivo, con el objetivo de impulsar una adecuada formación profesional. Este concepto de competencia profesional viene marcando la orientación de las iniciativas y procesos de cambio estratégicos que durante la última década están poniendo en marcha distintos países en torno a cuatro ejes de actuación [67]:

- El acercamiento entre el mundo laboral y a formación
- La adecuación de los profesionales a los cambios en la tecnología y en las organizaciones
- La renovación de las entidades de educación, de los equipos docente y de la propia oferta educativa
- Las modalidades de adquisición y reconocimiento de las calificaciones.

En [4] se puede apreciar textualmente que "Frente a los modelos educativos eclécticos y frente a la desvinculación de la vida académica de los requerimientos del reclutamiento, la selección, el entrenamiento y la evaluación del desempeño para el trabajo; con el creciente número de inferencias que se requiere para, con dificultad, observar y evaluar el dominio de lo aprendido y el resultado del aprendizaje, ha surgido y se ha desarrollado crecientemente, a lo largo de los últimos quince años, la educación basada en competencias".

Según la UNESCO [99], se entiende por competencia "La estrategia educativa que evidencie el aprendizaje de conocimientos, las capacidades, actitudes y comportamientos requeridos para desempeñar un papel específico, ejercer una profesión o llevar a cabo una tarea determinada. Concebidas también como una compleja estructura de atributos y tareas, que permiten que ocurran varias acciones intencionadas simultáneamente, es en el contexto (cultura y lugar) en el cual se llevan a cabo la acción. Incorporan la ética y los valores".

Aunque una competencia se entiende como algo integral que combina los diversos aspectos del individuo y de su entorno, el logro de las mismas no se puede medir en la forma tradicional pues su efecto se manifiesta en el desempeño del individuo en una actividad concreta. También, La formulación de objetivos específicos no debe entenderse como la formulación de competencias específicas pues estas son algo más amplio, sin embargo, dichos objetivos son una necesidad. La diferencia está en el accionar para el logro de los objetivos pues se debe buscar un desarrollo integral de habilidades, conocimientos y actitudes.

Según [67], las competencias profesionales se caracterizan porque integran un conjunto de conocimientos, procedimientos, actitudes y rasgos que se complementan entre si, de manera que la persona de “saber”, “saber hacer”, “saber estar” y “saber ser” para actuar con eficacia frente a situaciones profesionales. Las competencias solo son definibles en la acción. La combinación de habilidades, conocimientos y contexto dónde se desarrollan supone una revolución en los sistemas educativos [32].

De acuerdo a los modelos curriculares de la ANIEI [74] en México y al diseño de la Ingeniería Informática del Libro Blanco Español [67], la programación es una competencia específica que se evalúa en el contexto de la práctica.

Siendo la programación una competencia y las competencias estar integradas por habilidades, el proceso formativo de las competencias es completamente aplicable al de las habilidades. Sin embargo, estos procesos formativos históricamente no estaban orientados hacia la formación y desarrollo de habilidades y competencias. En las siguientes secciones se analizan los enfoques y modelos que en el área educativa se han aplicado.

3.6.3.1. Dos modelos de Educación

Desde que el hombre tuvo conciencia de si mismo como resultado de la evolución de millones de años que se coronan con el nacimiento del primer Homo Sapiens (o como ahora se le señala Sapiens Sapiens) hay un factor que ha dirigido su crecimiento, evolución y supervivencia en el mundo: su capacidad de aprender y utilizar ese conocimiento adquirido en sus acciones cotidianas para poder, sobrevivir primero y dominar después, este planeta.

Diversas teorías tratan de fundamentar el éxito del ser humano como especie. La que se ha posicionado como la más sensata elimina muchas casuísticas y factores externos de naturaleza ambiental y climática y se centra en tres características humanas claves para su éxito. Estas son: La capacidad mental, que incluye el aprendizaje y el razonamiento; La capacidad de comunicarse, que incluye tanto la estructuración de un lenguaje verbal y las capacidades fisiológicas para articular palabras; y la capacidad de crear y hacer, gracias a las manos. Conjuntado estos tres elementos La habilidad para comunicarse, la habilidad de hacer cosas con las manos y la habilidad mental para aprender y reflexionar se fundamenta el éxito del ser humano como especie.

Sobre esta plataforma conceptual se puede observar que uno de los factores que han dirigido al ser humano en su viaje por la historia es la capacidad de aprender y de utilizar este conocimiento

en su cotidiano quehacer. El estudio de esta habilidad, su evolución y como perfeccionarlo ha sido motivo de interés durante mucho tiempo.

En sus inicios la visión estaba centrada en el profesor y la enseñanza. Desde aquellas jornadas en los jardines de Academus en la antigua Grecia y hasta el siglo pasado la forma como los seres humanos se desarrollaban era siguiendo los pasos de un mentor, el cual debía contar con un derroche de habilidades, valores y destrezas para ser considerado un buen maestro.

Durante siglos las habilidades de los profesores se convirtieron en condición *sin equanon* para el éxito del aprendizaje de las personas. Era tal la convicción de esta premisa que se desarrollaron tratados completos sobre técnicas de enseñanza y siempre privilegiando lo que el profesor debe o no debe hacer en el ejercicio de su cátedra para lograr el aprendizaje de las personas.

El siguiente paradigma histórico fue reconocer que no bastaba con estudiar las técnicas de enseñanza sino que, poco a poco, el aprendizaje mismo comenzó a tener un papel más importante hasta llegar al punto de que se habla del “binomio” *enseñanza-aprendizaje* en el cual es igualmente importante la enseñanza como el aprendizaje en el más puro principio de una balanza equilibrada. Sin embargo, se sigue considerando al profesor como el principal actor y responsable para que ese binomio tomara vida. Este paradigma se le ha llamado coloquialmente “educación tradicional” o “conductista” ya que principalmente aplica una estrategia inductiva.

Existen varias premisas importantes que son parte del modelo tradicional:

- El profesor es el actor principal del acto educativo.
- El éxito en el proceso educativo depende enteramente de las habilidades del docente.
- El alumno debe ser disciplinado, obediente, silencioso y mantener constantemente la atención en lo que el profesor explica (debe ser pasivo mientras el conocimiento se le transmite).
- La técnica por excelencia en el modelo es la exposición verbal, por parte del docente, de los conocimientos que el alumno debe adquirir (inducción).

Como toda ciencia, la educación no puede permanecer inmóvil. Nuevas teorías y propuestas han ido surgiendo con el tiempo. De hecho, sucedió que cada día el concepto del aprendizaje tomaba más y más importancia, ganando peso en la “balanza” hasta un punto en el cual se convirtió en el foco de atención prioritario y el modelo se transformó en lo que actualmente se le llama “modelo centrado en el aprendizaje”.

El modelo centrado en el aprendizaje se basa en las siguientes premisas:

- El alumno es el actor principal del acto educativo.
- El éxito depende del compromiso del estudiante para con su propio desarrollo, el profesor es un detonante de este compromiso.
- El alumno debe ser pro-activo, inquieto, crítico, reflexivo y mantenerse en constante acción en la búsqueda del conocimiento.
- El profesor toma el rol de un facilitador que orienta sobre las diversas formas de enfrentar, analizar y resolver un problema acercando los conocimientos al estudiante.

- Las técnicas se basan principalmente en métodos deductivos e intuitivos, basadas en el descubrimiento e incorporación del conocimiento dentro de un marco completamente contextualizado a la realidad del alumno y poder generar aprendizaje significativo.

Hoy en día ha quedado demostrado que lo más importante no es como enseñan los profesores sino como aprenden los alumnos. Se han elaborado diversas propuestas sobre los estilos de aprendizaje de las personas; como lograr potenciar la memoria; como la reflexión y relaciones entre los conceptos permiten una mejor comprensión de las cosas; como la participación activa y grupal de los estudiantes favorecen el aprendizaje; como los ambientes lúdicos y relajados crean una buena disposición para aprender mejor; e incluso, como aprender a aprender.

Sin embargo, y a pesar de todo lo comentado, los cambios de paradigma nunca han sido fáciles. Siempre se ha derramado sangre, sudor y lágrimas para permitir que las nuevas ideas tomen su lugar en el concierto de la filología universal. El principal problema al que se enfrenta el educador y educando modernos que quieren vivir en este nuevo modelo es el Shock Cultural del Aprendizaje. El cual no es más que la reacción provocada por la inercia del modelo tradicional vivido durante años por los estudiantes (desde niños hasta adolescentes) y de los profesores (que toda su vida han realizado actividades para enseñar y no para aprender). Este shock lo viven de forma diferente el profesor y el alumno. El profesor tiene temor a renunciar a ser el actor principal y conductor del acto de aprendizaje y se cuestiona sobre si es en verdad posible que un estudiante pueda construir el conocimiento sin que él se lo diga; por parte del alumno, tiene el temor de hacerse responsable de su propio conocimiento, siempre es más fácil que sea otro el que haga las cosas.

La coyuntura educativa en la cual se vive actualmente exige visualizar, prevenir, enfrentar y sobreponerse a este Shock Cultural del Aprendizaje con un profundo sentido de responsabilidad y compromiso. El único camino, paradójicamente, es la educación misma. Se tiene que desaprender (alumnos y profesores) como se hacían las cosas antes para poder renacer en un nuevo y fresco escenario colaborativo, centrado en el aprendizaje y en el alumno; dónde el pasado no determine el futuro, sino que éste sea dictado por la esperanza de vida que cada uno de los estudiantes deposita en las manos de sus maestros.

3.6.3.2. Construyendo el Conocimiento

El concepto de construcción del conocimiento proviene del creciente enfoque de privilegiar el aprendizaje sobre otros factores y como resultado natural de los estudios que se han venido realizando desde hace muchos años sobre el fenómeno del aprendizaje.

El constructivismo sostiene que una persona aprende la realidad del mundo a través de establecer abstracciones (modelado) de la misma y que éstas se construyen en forma completamente individual y personal en función de las experiencias que a lo largo de la vida esta persona ha vivido. Cada aprendizaje nuevo se asocia con los conocimientos previos del mundo (que en realidad es su propia construcción de una representación de éste) formando una plataforma que se enriquece día a día con los nuevos conocimientos experiencias.

Las ideas principales que el constructivismo defiende son las siguientes:

1. El alumno es el responsable de su propio proceso de aprendizaje. Dado que el aprendizaje es una construcción mental estrictamente personal e individual nadie puede hacer que otro aprenda si éste no lo desea o no hace el esfuerzo de hacerlo engarzando las nuevas ideas con las que ya tenía previamente.
2. La actividad mental constructiva del alumno se basa en los contenidos que previamente tiene construidos.
3. El hecho de que el proceso constructivo está en relación con las construcciones mentales previas del alumno condiciona la labor que el facilitador debe realizar la cual ya no es enseñar sino dejar aprender.

El proceso de construcción del conocimiento no es fortuito ni casual, no se da al azar ni mucho menos depende del profesor. El proceso de construcción del conocimiento es intencional, voluntario y deseado. Pero sobre todo, implica atribuirle un significado, darle una representación mental de lo que se desea aprender. Para que la construcción del conocimiento pueda darse hay una serie de condiciones que deben existir para poder, a continuación seguir un proceso de construcción.

Condiciones:

- Debe haber un motivo para aprender. La gente no aprende sólo porque sí o porque “tiene que”, sino que aprende con una intención. Esta intención es el motivador principal para aprender. Muchos pueden ser los motivos, entre ellos puede estar el hecho de que sea interesante, relevante, importante, ansiado, conveniente, en fin. Es tarea del facilitador encontrar cuales son los motivos por los cuales una persona puede querer aprender algo e incluso crearlos cuando no los hay.
- Todo aprendizaje tiene una razón de ser. Estos motivos deben estar relacionados con su entorno y su realidad para poder vivenciarlos más fácilmente. En palabras simples debe haber una disposición favorable hacia el aprendizaje.
- El contenido debe ser valioso y potencialmente significativo para cumplir con la motivación que se tiene. Una vez que se han identificado los motivos por los cuales una persona desea aprender algo, los contenidos que se le presentan deben estar relacionados con esos motivos o se perderá el significado mismo del aprendizaje. El aprendizaje será significativo si estas dos condiciones convergen.

Proceso de construcción del aprendizaje:

1. Descubrir o crear la necesidad del aprendizaje.
2. Desarrollar o retomar el sentido de responsabilidad personal con respecto a los conocimientos que se requieren para satisfacer la necesidad.
3. Estructurar la experiencia de tal forma que tenga una aplicación práctica y esté vinculada con los conocimientos previos. En otras palabras crear relaciones y asociaciones con el contexto.
4. Vivir la experiencia buscando y aplicando el aprendizaje. Es importante poder demostrar lo aprendido y generar confianza.

5. Reconocer, estimular y aprobar lo aprendido
6. Iniciar el ciclo de nuevo.

El proceso de aprendizaje es cíclico en una espiral ascendente que cada día integra más y nuevos conocimientos basados en los que ya se tienen. Esta espiral, como todo fenómeno físico necesita de impulso para conservar su fuerza o de lo contrario se frena. Las dos condiciones señaladas pueden ser explotadas por el facilitador para que se conviertan en un motor permanente. La misión última del facilitador es poder crear tal impulso que eventualmente el motor de la motivación pueda ser alimentado por el propio estudiante sin necesidad del profesor. Si se logra esta magia de que el alumno se encuentre automotivado, dispuesto y comprometido con su aprendizaje se puede pensar que habrá éxito en el proceso.

3.6.4. Taxonomía de Habilidades

La competencia de la programación se desglosa en una serie de habilidades técnicas, las cuales a su vez se componen de habilidades intelectuales. La metodología de microingeniería, componente de la dimensión teórica y explicada en la sección 3.4, está dividida en varias fases las cuales cada una requiere de distintas habilidades técnicas. La tabla 26 es una clasificación de las habilidades técnicas a través del modelo de Guilford. Conociendo cual es la habilidad intelectual a la que corresponde cada habilidad técnica es posible utilizar las estrategias comunes de desarrollo de habilidades.

FASE/ACTIVIDAD	HABILIDADES TÉCNICAS	HABILIDADES INTELECTUALES
1ª ITERACIÓN		
Fase de análisis (1ª Iteración).		
Lectura de comprensión del problema	Identificar las partes del problema	CUM, CRM, CSM, MUF, NTS
Identificación de los resultados	Identificación de las salidas requeridas por el problema	CRS, CRM, ERS, ERM, CCS
Identificación de los procesos	Creación de árbol de descomposición de procesos	NRS, NRM, DSS, DSM, NCS, DCS
	Definición y evaluación de formulas para la obtención de las salidas	ERS, ERM, ESS, ESM, NSS, CIS, MCS, NCS
	Traducir formulas de notación algebraica a algorítmica y viceversa	ETS, ETM, NTS, NTM, CIS, MCS, NCS, DUS
	Aplicación de precedencia de operadores aritméticos	MUS, MRS, ERS, ERM, CIS, MCS, NCS
Identificación de datos de entrada	Reconocer las entradas necesarias para los procesos identificados	CUM, CRM, ERS, ERM, CCS
Definición de Diccionario de Datos	Definición de tipos de datos	CCS, CCM, CUS, CUM
	Definición de dominios	ECM, ERM, ECS, ERS, CIS
	Definición del destino de los datos	ESS, ESM
Fase de diseño (1ª Iteración).		
Diagramas de secuencia	Definición de secuencias de pasos lineales	NSS, NSM, NTS, NTM, DIM, DRS, MIS, NCS, DCS
Integración	Concatenación de dos o más secuencias	NTS, NTM, NIS, NIM, NCS
	Fusión de dos o más secuencias	NTS, NTM, NIS, NIM, NCS
	Anidación de dos o más secuencias	NTS, NTM, NIS, NIM, NCS
	Adaptación de dos o más secuencias	DTS, DTM, DIS, DIM, DRS, MIS, NCS

FASE/ACTIVIDAD	HABILIDADES TÉCNICAS	HABILIDADES INTELECTUALES
Refinamiento	Identificación y aplicación de mejoras a procesos ya definidos.	DTS, DTM, DIS, DIM, DRS, MIS, MTS
Diseño de pruebas	Definición de conjuntos validos e inválidos de datos	NSM, NIM, DSM, DIM
Fase de construcción (1ª Iteración).		
Codificación del diseño integrado y refinado en lenguaje 3GL	Traducción de lenguaje simbólico a lenguaje de programación	ETS, ETM, NTS, NTM, CTS, MTS, DUS
Fase de pruebas (1ª Iteración).		
Aplicación de pruebas	Evaluación de procedimientos secuenciales	ESS, ESM, ERS, ERM
	Identificación de errores en secuencias	EIM, EIS, NIS, NIM
	Solución de errores de secuencias	NIS, NIM, DIS, DIM
Fase de liberación (1ª Iteración).		
Integración de productos	Organización documental	MRS, MRM, MSS, MSM
	Organización de versiones de producto de software	MRS, MRM, MSS, MSM
2ª ITERACIÓN.		
Fase de análisis (2ª Iteración).		
Identificación de Restricciones	Definición de dominios de datos de entrada	ERS, ERM, ESS, ESM, NSS, CIS
	Identificación de restricciones del dominio del problema	CRS, CRM, ERS, ERM, EIM
	Identificación de restricciones del dominio de la solución	CRS, CRM, ERS, ERM, EIM
	Aplicación de precedencia de operadores relacionales	MUS, MRS, ERS, ERM, CIS, MCS, NCS
	Diseño de condiciones simples	NRS, NRM, NIS, NIM, DCS
	Evaluación de condiciones simples	ERS, ERM, EIS, EIM
	Aplicación de precedencia de operadores lógicos	MUS, MRS, ERS, ERM, CIS, MCS, NCS
	Diseño de condiciones compuestas	NRS, NRM, NIS, NIM, DCS
	Evaluación de condiciones complejas	ERS, ERM, EIS, EIM
	Aplicación de precedencia de operadores relaciones	MUS, MRS, ERS, ERM, CIS, MCS, NCS
	Diseño de árboles de decisiones	NIS, CIM, DIS, DIM, DCS
	Diseño de tablas de decisiones	NIS, CIM, DIS, DIM, DCS
	Aplicación de Leyes de Morgan	MRS, MRM, ERS, ERM, CIS, MCS, NCS
Fase de diseño (2ª Iteración).		
Diseño de restricciones	Diseño de estructuras condicionales simples	NSS, NSM, NTS, NTM, DIM, DRS, MIS, DCS
	Diseño de estructuras condicionales dobles	NSS, NSM, NTS, NTM, DIM, DRS, MIS, DCS
	Diseño de multi-condicionales	NSS, NSM, NTS, NTM, DIM, DRS, MIS, DCS
Integración	Concatenación de dos o más estructuras condicionales	NTS, NTM, NIS, NIM, NCS
	Fusión de dos o más secuencias	NTS, NTM, NIS, NIM, NCS
	Anidación de dos o más estructuras condicionales	NTS, NTM, NIS, NIM, NCS
	Adaptación de dos o más estructuras condicionales	DTS, DTM, DIS, DIM, DRS, MIS, NCS

FASE/ACTIVIDAD	HABILIDADES TÉCNICAS	HABILIDADES INTELECTUALES
Refinamiento	Identificación de mejoras a los procesos ya definidos.	DTS, DTM, DIS, DIM, DRS, MIS, MTS
Diseño de pruebas	Definición de conjuntos validos e inválidos de datos para cada posible ruta de ejecución	NSM, NIM, DSM, DIM
Fase de construcción (2ª Iteración).		
Codificación del diseño integrado y refinado en lenguaje 3GL	Traducción de lenguaje simbólico a lenguaje de programación	ETS, ETM, NTS, NTM, CTS, MTS, DUS
Fase de pruebas (2ª Iteración).		
Aplicación de pruebas	Evaluación de procedimientos con estructuras condicionales	ESS, ESM, ERS, ERM
	Identificación de errores en estructuras condicionales	EIM, EIS, NIS NIM
	Solución de errores de estructuras condicionales	NIS, NIM, DIS, DIM
Fase de Liberación (2ª Iteración).		
Integración de productos	Organización documental	MRS, MRM, MSS, MSM
	Organización de versiones de producto de software	MRS, MRM, MSS, MSM
3ª ITERACIÓN.		
Fase de Análisis (3ª Iteración).		
Identificación de repeticiones	Identificación y definición de repeticiones cerradas	CRS, CRM, ERS, ERM, EIM
	Identificación y definición de repeticiones abiertas con condición al inicio del proceso	CRS, CRM, ERS, ERM, EIM
	Identificación y definición de repeticiones abiertas con condición al final del proceso	CRS, CRM, ERS, ERM, EIM
	Identificación de requerimientos de conteo	CSS, CSM, ESS, ESM, EIM
	Identificación de requerimientos de acumulación	CSS, CSM, ESS, ESM, EIM
	Identificación de requerimientos de permutación	CSS, CSM, ESS, ESM, EIM
Fase de diseño (3ª Iteración).		
Diseño de repeticiones	Diseño de estructuras cíclicas cerradas	NSS, NSM, NTS, NTM, DIM, DRS, MIS, DCS
	Diseño de estructuras cíclicas abiertas con condición al inicio de la estructura	NSS, NSM, NTS, NTM, DIM, DRS, MIS, DCS
	Diseño de estructuras cíclicas abiertas con condición al final de la estructura	NSS, NSM, NTS, NTM, DIM, DRS, MIS, DCS
	Diseño de contadores	NSS, NSM, NTS, NTM, DIM
	Diseño de sumatorias	NSS, NSM, NTS, NTM, DIM
	Diseño de permutaciones	NSS, NSM, NTS, NTM, DIM
Integración	Concatenación de dos o más estructuras cíclicas	NTS, NTM, NIS, NIM, NCS
	Fusión de dos o más cíclicas	NTS, NTM, NIS, NIM, NCS
	Anidación de dos o más estructuras cíclicas	NTS, NTM, NIS, NIM, NCS
	Adaptación de dos o más estructuras cíclicas	DTS, DTM, DIS, DIM, DRS, MIS, NCS
Refinamiento	Identificación de mejoras a los	DTS, DTM, DIS, DIM, DRS, MIS,

FASE/ACTIVIDAD	HABILIDADES TÉCNICAS	HABILIDADES INTELECTUALES
	procesos ya definidos.	MTS
Diseño de pruebas	Definición de conjunto de datos de control de ciclo cerrado	NSM, NIM, DSM, DIM
	Definición de conjunto de datos de control de ciclo abierto con posibilidad de no realizar ni siquiera una iteración	NSM, NIM, DSM, DIM
	Definición de conjunto de datos de control de ciclo abierto con al menos una repetición.	NSM, NIM, DSM, DIM
Fase de construcción (3ª Iteración).		
Codificación del diseño integrado y refinado en lenguaje 3GL	Traducción de lenguaje simbólico a lenguaje de programación	ETS, ETM, NTS, NTM, CTS, MTS, DUS
Fase de pruebas (3ª Iteración).		
Aplicación de pruebas	Evaluación de procedimientos con estructuras cíclicas	ESS, ESM, ERS, ERM
	Identificación de errores en estructuras cíclicas	EIM, EIS, NIS, NIM
	Solución de errores de estructuras cíclicas	NIS, NIM, DIS, DIM
Fase de liberación (3ª Iteración).		
Integración de productos	Organización documental	MRS, MRM, MSS, MSM
	Organización de versiones de producto de software	MRS, MRM, MSS, MSM
FASE DE INTEGRACIÓN DE PRODUCTO FINAL		
Liberación de versión 1.0	Organización de versiones de producto de software	MRS, MRM, MSS, MSM
Integración de productos documentales	Organización documental	MRS, MRM, MSS, MSM
OTRAS ACTIVIDADES		
Mantenimiento de código propio	Lectura e interpretación de código propio	CUM, CRM, CSM, MUF, NTS
Mantenimiento de código ajeno	Lectura e interpretación de código ajeno	CUM, CRM, CSM, MUF, NTS
Integración de código desarrollado en grupos de trabajo	Adaptación de dos o más procedimientos integrados	DTS, DTM, DIS, DIM
Ingeniería Inversa	Descubrimiento y elaboración del diseño a partir de lectura de código	NSM, NIM, ESM, EIM
Reutilización de código	Identificación de código común	CSS, CSM, ESS, ESM
	Transformación de estructuras para reutilización de código	ETS, ETM, EIS, EIM
Traducción entre lenguajes	Traducción de un lenguaje de programación a otro	ETS, ETM, NTS, NTM
Elaboración de código a partir de diseños de terceros	Lectura e interpretación de diseño.	ESS, ESM, CSS, CSM
	Transcripción de diseño a código 3GL:	ETS, ETM, NTS, NTM

Tabla 26. Habilidades intelectuales que corresponden a las técnicas

La tabla 26 permite hacer un mapa entre las habilidades técnicas requeridas en cada fase de la metodología y las habilidades intelectuales. En la tabla 27 se muestra un conteo estadístico de las habilidades intelectuales requeridas por etapa.

Clave	Descripción	Análisis	Diseño	Codif.	Pruebas	Liber.	Total
DIM	Inferencias semánticas	2	21	0	3	0	26
NTS	Transformaciones simbólicas	2	19	3	0	0	24
NTM	Transformaciones verbales	1	19	3	0	0	23
NIM	Implicaciones semántica	2	14	0	6	0	22
NCS	Clasificación simbólica	8	13	0	0	0	21
ERM	Eval. conceptos que se relacionan	17	0	0	3	0	20
ERS	Análisis relaciones simbólicas	17	0	0	3	0	20
NIS	Solución problemas simbólicos	4	9	0	6	0	19
NSM	Reproducción sistemas semánticos	0	15	0	0	0	15
DRS	Relaciones simbólicas	0	13	0	0	0	13
EIM	Análisis implicaciones semánticas	10	0	0	3	0	13
MIS	Implicaciones simbólicas	0	13	0	0	0	13
DCS	Conceptualización simbólica	5	7	0	0	0	12
NSS	Aplicación sistemas simbólicos	2	10	0	0	0	12
DIS	Inferencias simbólicas	2	6	0	3	0	11
MRS	Relaciones simbólicas	5	0	0	0	6	11
CIS	Implicaciones simbólicas	9	0	0	0	0	9
ESM	Eval. secuencias semánticas	6	0	0	3	0	9
ESS	Evaluación sistemas simbólicos	6	0	0	3	0	9
CRM	Razonamiento analógico	8	0	0	0	0	8
MCS	Conceptos simbólicos	7	0	0	0	0	7
MRM	Relaciones semánticas	1	0	0	0	6	7
CRS	Relaciones numéricas	6	0	0	0	0	6
DSM	Sistematización semántica	1	5	0	0	0	6
DTM	Transformaciones verbales	0	6	0	0	0	6
DTS	Transformaciones simbólicas	0	6	0	0	0	6
MSM	Secuencias verbales	0	0	0	0	6	6
MSS	Secuencias simbólicas	0	0	0	0	6	6
MTF	Evocac. transformaciones fig.	0	3	3	0	0	6
EIS	Análisis implicaciones Simbólicas	2	0	0	3	0	5
CSM	Comprensión verbal	4	0	0	0	0	4
DUS	Fluidez simbólica	1	0	3	0	0	4
ETM	Transformaciones semánticas	1	0	3	0	0	4
ETS	Discriminación transform. simbólicas	1	0	3	0	0	4
MUS	Evocación detalles simbólicos	4	0	0	0	0	4
CCS	Conceptos numéricos	3	0	0	0	0	3
CSS	Sistemas numéricos	3	0	0	0	0	3
CTS	Transformaciones simbólicas	0	0	3	0	0	3
CUM	Vocabulario	3	0	0	0	0	3
NRM	Aplicación razonamiento analógico	3	0	0	0	0	3
NRS	Aplicación relaciones simbólicas	3	0	0	0	0	3

Clave	Descripción	Análisis	Diseño	Codif.	Pruebas	Liber.	Total
CIM	Implicaciones verbales	2	0	0	0	0	2
CCM	Conceptualización	1	0	0	0	0	1
CUS	Reconocimiento simbólico	1	0	0	0	0	1
DSS	Sistematización simbólica	1	0	0	0	0	1
ECM	Discriminación conceptual	1	0	0	0	0	1
ECS	Discriminación concep. simbólicos	1	0	0	0	0	1
MUF	Evocación detalles figurativos	1	0	0	0	0	1
TOTALES	49 Habilidades	157	179	21	36	24	417

Tabla 27. Habilidades intelectuales requeridas por fase

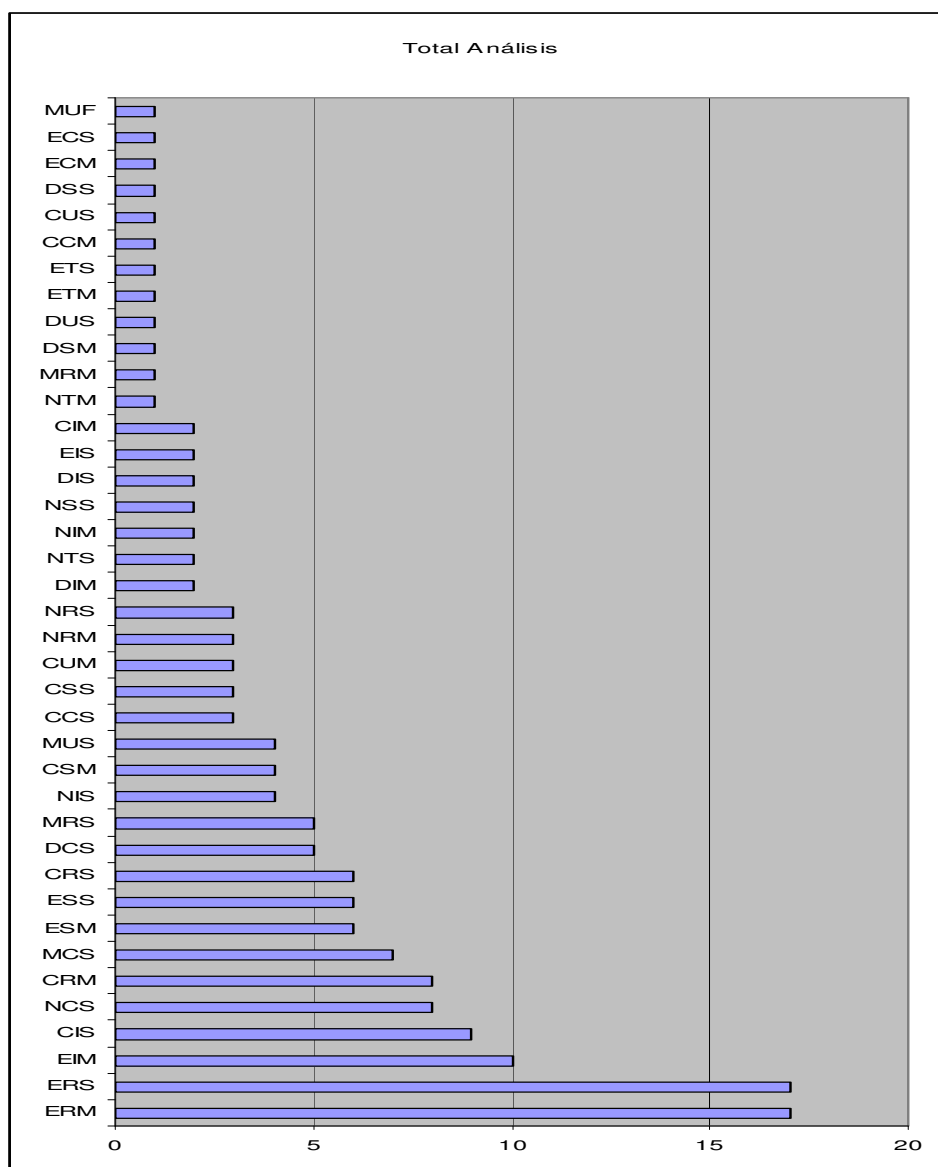


Figura 71. Habilidades intelectuales en el análisis

Como puede observarse, en la competencia de la programación se aplican 49 habilidades intelectuales. Algunas habilidades se repiten en diferentes fases de la metodología lo que

significa que son de más impacto que otras. Por ejemplo, en la fase de análisis las habilidades más reiteradas son ERM (Evaluación de Relaciones Semánticas) y ERS (Evaluación de Relaciones Simbólicas). Estas habilidades resultan cruciales en esta etapa y su desarrollo y fortalecimiento es de alto impacto. En la figura 71 se observa el total de habilidades que afectan la fase de análisis.

En la figura 72 se muestran las habilidades involucradas en la fase de diseño. Como puede apreciarse, en el análisis se aplican 39 habilidades intelectuales mientras que en el diseño únicamente 16.

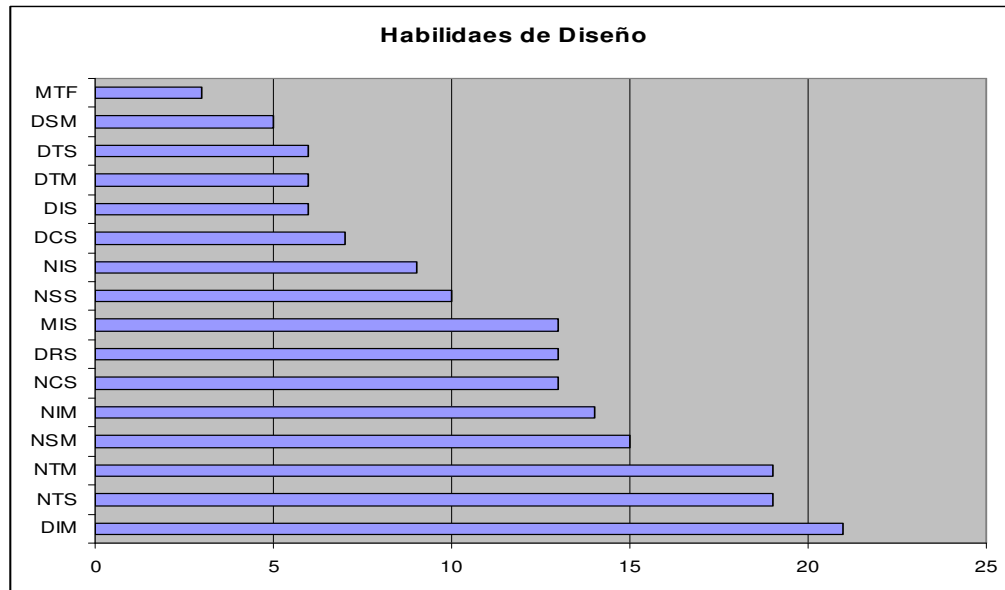


Figura 72. Habilidades intelectuales en el diseño

En la fase de codificación solamente participan siete habilidades todas ellas con un impacto de tres. Estas habilidades son: NTS, NTM, MTF, DUS, ETM, ETS y CTS. En lo que corresponde a la fase de pruebas se muestra en la figura 73 la gráfica correspondiente, en la que se aprecia que son 10 las habilidades intelectuales en operación.

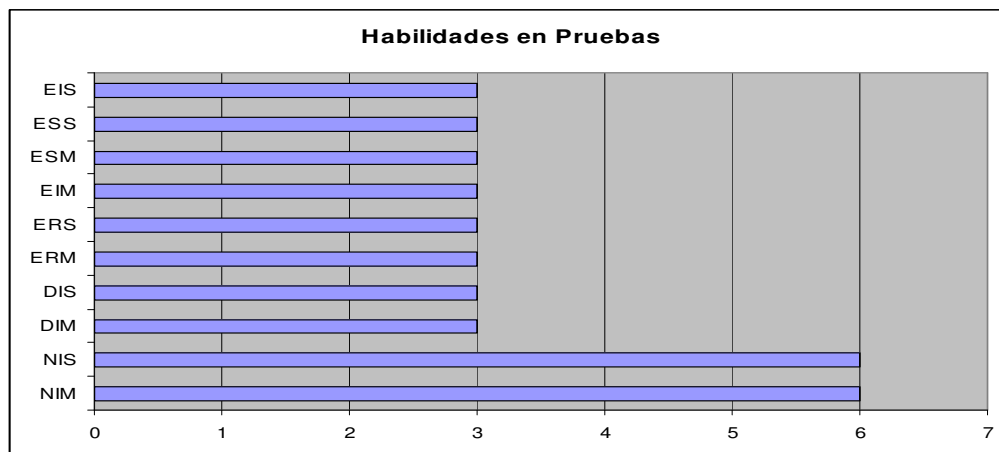


Figura 73. Habilidades intelectuales en la etapa de pruebas

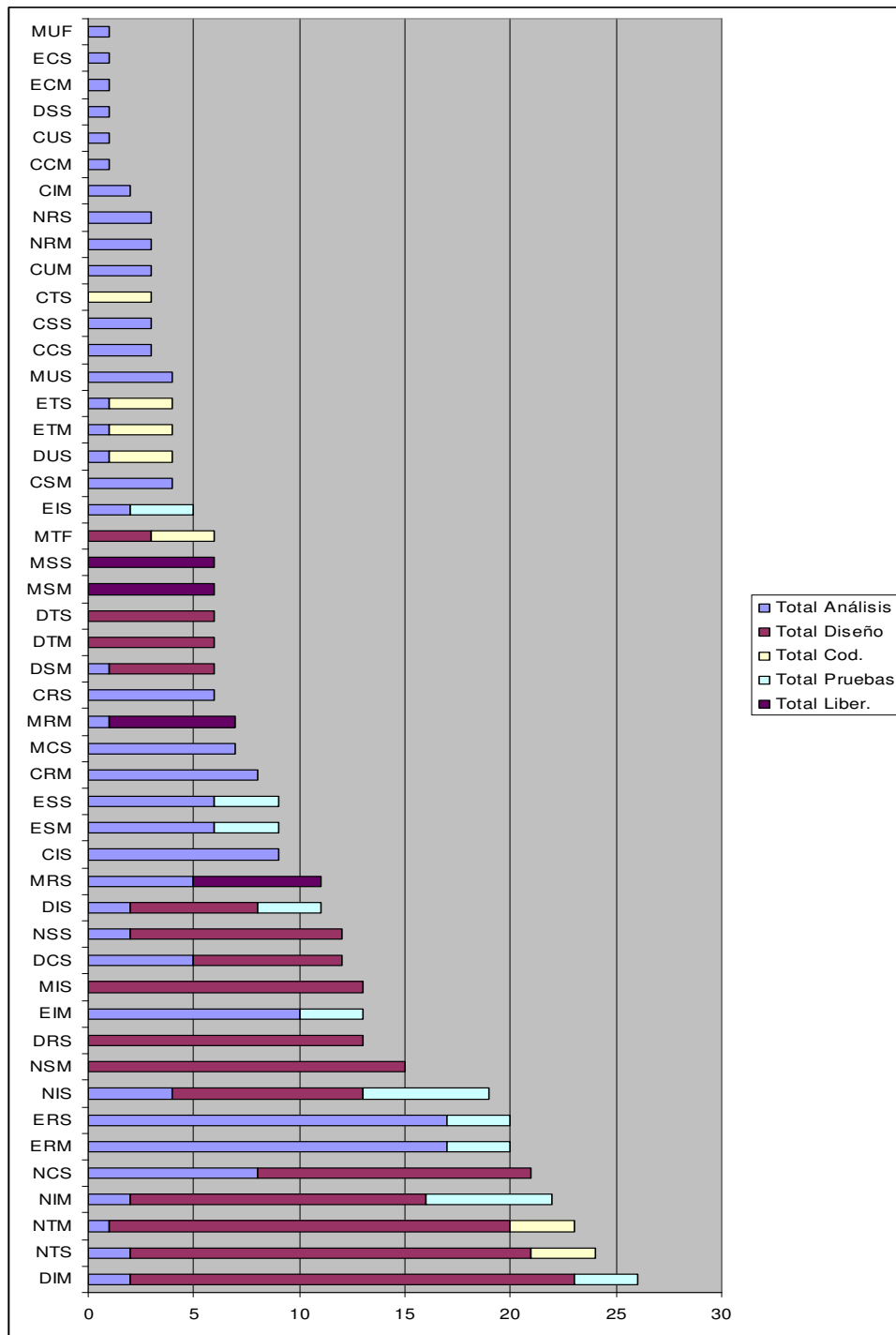


Figura 74. Habilidades en la programación

Finalmente, en la fase de liberación participan cuatro habilidades, todas ellas repetidas en seis ocasiones y son MRS, MRM, MSM y MSS.

En la figura 74 se encuentran integradas todas las habilidades intelectuales requeridas en la programación y se muestra en cada barra en que fase del ciclo de vida se aplican.

La figura 75 muestra que el 52% de las habilidades se aplican en la fase de análisis, el 21% en diseño, solo el 9% en codificación, el 13% en la fase de pruebas y el 5% en la liberación. Resulta claro que la fase más relevante y de alto impacto es la fase de análisis.

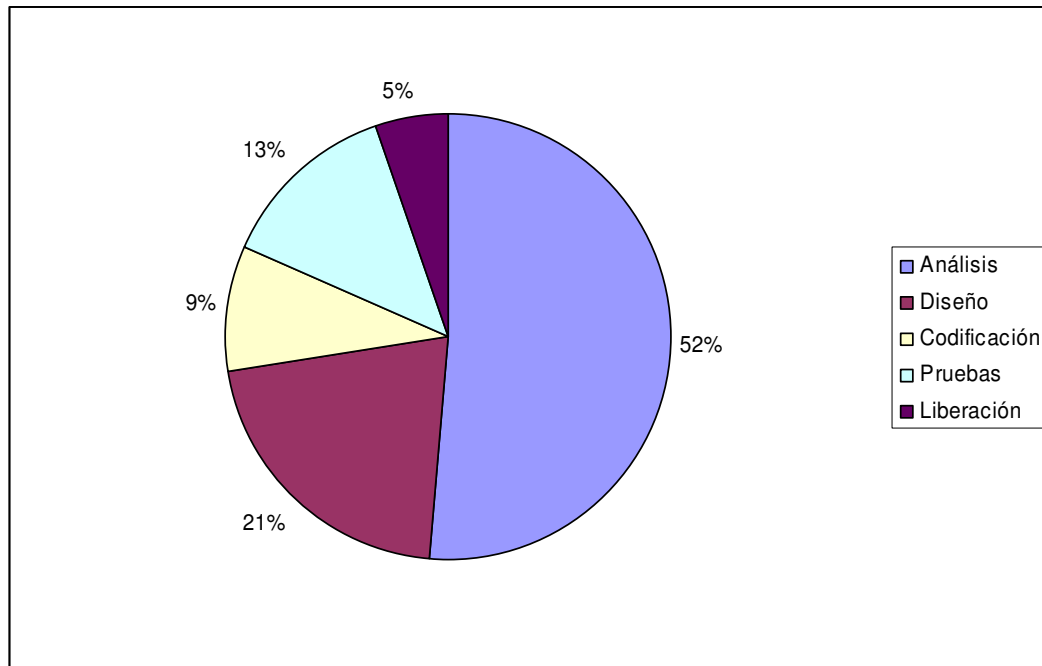


Figura 75. Proporción de habilidades por fase

De acuerdo a un estudio publicado por la Universidad de Buenos Aires, Argentina [27]; realizado entre 1997 y 2004 han encontrado que el 30% de los estudiantes que no acreditan la materia de programación se debe a que tienen una mala interpretación del problema a resolver. Este error se comete en la fase de análisis. Si se puede desarrollar las habilidades intelectuales que impactan en ella se podrá mejorar en mucho la competencia de la programación. En la tabla 28 se muestra el resumen de resultados del estudio citado.

Tipo Error	Error global	Items
1	Errores debidos a mala interpretación del problema a resolver.	30.0%
2	Errores debido a procesos de decisión mal elaborados.	20.8%
3	Errores debidos a mal manejo de las estructuras básicas de programación y de datos.	17.0%
4	Errores debidos a deficiencias en el uso del código de programación.	15.0%
5	Errores en el uso de variables globales y locales, pasaje de parámetros.	12.6%

Tabla 28. Porcentaje de errores en la programación

3.6.5. Desarrollando las Habilidades para la Programación

El modelo de Guilford aporta un mecanismo de identificación de las habilidades, para desarrollarlas se utilizan estrategias de fortalecimiento de las mismas. Dado que cada habilidad está fundamentada en sus predecesoras resulta ocioso redactar para todas y cada una acciones específicas.

Debido a que las habilidades se pueden desarrollar y fortalecer con la práctica y estimulación es importante abordar los principios de los estilos que cada persona utiliza para esta función. Esto es conocido como los Estilos de Aprendizaje.

3.6.5.1 Estilos de Aprendizaje

Una definición de estilos de aprendizaje dada por Keefe en [22]: “los estilos de aprendizaje son los rasgos cognitivos, afectivos y fisiológicos que sirven como indicadores relativamente estables, de cómo los alumnos perciben interacciones y responden a sus ambientes de aprendizaje”.

Los rasgos cognitivos tienen que ver con la forma en que los estudiantes estructuran los contenidos, forman y utilizan conceptos, interpretan la información, resuelven los problemas, seleccionan medios de representación (visual, auditivo, kinestésico), etc. Los rasgos afectivos se vinculan con las motivaciones y expectativas que influyen en el aprendizaje, mientras que los rasgos fisiológicos están relacionados con el biotipo y el biorritmo del estudiante.

El término ‘estilo de aprendizaje’ se refiere al hecho de que cada persona utiliza su propio método o estrategias a la hora de aprender. Aunque las estrategias varían según lo que se quiera aprender, cada uno tiende a desarrollar ciertas preferencias o tendencias globales, tendencias que definen un estilo de aprendizaje. Se habla de una tendencia general, puesto que, por ejemplo, alguien que casi siempre es auditivo puede en ciertos casos utilizar estrategias visuales.

Los estilos de aprendizaje se utilizan para recomendar estrategias de aprendizaje y desarrollo de habilidades, no son convenientes para clasificar a las personas puesto que la manera de aprender evoluciona y cambia constantemente.

Revilla en [89] destaca algunas características de los estilos de aprendizaje señalando que son relativamente estables, aunque pueden cambiar; pueden ser diferentes en situaciones distintas; son susceptibles de mejorarse; y cuando a las personas se les enseña según su propio estilo de aprendizaje, aprenden y desarrollan sus habilidades con más efectividad.

3.6.5.2. Modelos de Estilos de Aprendizaje

Los distintos modelos y teorías existentes sobre estilos de aprendizaje ofrecen un marco conceptual que permite entender el comportamiento de las personas, como se relacionan con la

forma en que están aprendiendo y desarrollando sus habilidades y el tipo de acción que pueden resultar más eficaz en un momento dado.

Existe una diversidad de concepciones teóricas que han abordado, explícita o implícitamente, los diferentes ‘estilos de aprendizaje’. Por ejemplo, Kolb [22] se refiere a los estilos activo, reflexivo, teórico y pragmático. Otros tienen en cuenta los canales de ingreso de la información. En este último sentido se consideran los estilos visual, auditivo y kinestésico, siendo el marco de referencia, en este caso, la Programación Neurolingüística, una técnica que comprende las tres vías de acceso a la información: visual, auditiva y táctil [81]. Otro modelo aborda el concepto desde el enfoque de los mecanismos de procesamiento de la información proponiendo los estilos lógico y holístico.

En un intento de clasificar las diferentes teorías sobre estilos de aprendizaje se hace a partir del criterio que distingue entre selección de la información (estilos visual, auditivo y kinestésico), procesamiento de la información (estilos lógico y holístico), y forma de empleo de la información (estilos activo, reflexivo, teórico y pragmático).

Debe tenerse presente que en la práctica esos tres procesos están muy vinculados. Por ejemplo, el hecho de seleccionar la información visualmente, ello afectará la manera de organizarla o procesarla.

Otros modelos han enfatizado el tipo de inteligencia de acuerdo a la concepción de inteligencias múltiples de Gardner, y en algunas se tuvo en cuenta la dominación cerebral de acuerdo al modelo Herrmann (cuadrantes cortical izquierdo y derecho, y límbico izquierdo y derecho).

Otro modelo es el de Felder y Silverman [23], considera cuatro categorías donde cada una se extiende entre dos polos opuestos: activo/reflexivo, sensorial/intuitivo, visual/verbal y secuencial/global. Como puede advertirse, este es un modelo mixto que incluye algunos estilos de aprendizaje de otros modelos ya descriptos.

En la tabla 29 se muestra un resumen de los distintos modelos de estilos de aprendizaje:

Criterio de clasificación	Estilos de aprendizaje
Según el hemisferio cerebral	Lógico Holístico
Según el cuadrante cerebral (Herrmann)	Cortical izquierdo Límbico izquierdo Límbico derecho Cortical derecho
Según el sistema de representación (PNL)	Visual Auditivo Kinestésico
Según el modo de procesar la información (Kolb)	Activo Reflexivo Pragmático Teórico
Según la categoría bipolar (Felder y Silverman)	Activo/reflexivo

Criterio de clasificación	Estilos de aprendizaje
Silverman)	Sensorial/intuitivo Visual/verbal Secuencial/global
Según el tipo de inteligencia (Gardner)	Lógico-matemático Lingüístico-verbal Corporal-kinestésico Espacial Musical Interpersonal Intrapersonal Naturalista

Tabla 29. Estilos de aprendizaje

3.6.5.3. Estrategia de Desarrollo de Habilidades de Programación

Ya han sido explicados los conceptos de habilidad, inteligencia, modelos de aprendizaje, estilos de aprendizaje así como una taxonomía específica de habilidades para la programación. Lo siguiente es poder utilizar esta información para crear una estrategia de desarrollo de habilidades mentales que puedan facilitar el desarrollo de la competencia de la programación. Esta estrategia esta regida por los siguientes principios:

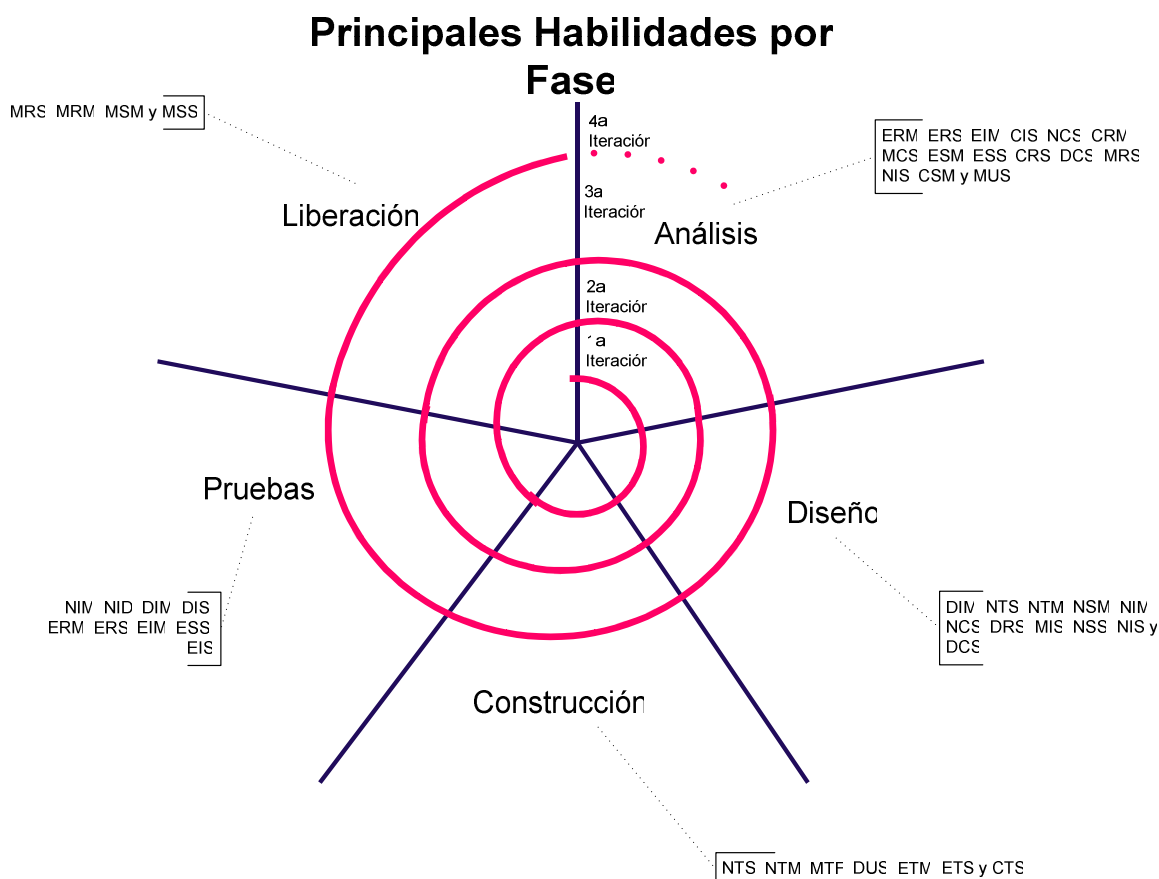


Figura 76. Modelo incremental de habilidades por fase

1. Para que una habilidad técnica sea desarrollada, las habilidades intelectuales tienen que estar desarrolladas.
2. Las habilidades intelectuales se desarrollan progresivamente y cada una depende de la fortaleza de sus predecesoras, las habilidades técnicas también.
3. Las habilidades se desarrollan con juegos, ejercicios y prácticas utilizando el modelo proximal de Vigotsky y contextualizadas a la realidad personal de quien las ejercita.
4. Las habilidades se desarrollan mejor si los juegos, ejercicios o prácticas aplicados corresponden al estilo de aprendizaje preferente para cada persona.
5. Las habilidades de la programación deben ser desarrolladas progresivamente de acuerdo a la espiral incremental de la metodología de microingeniería.

Siguiendo estos principios, la figura 76 muestra la propuesta de habilidades a fortalecer para cada fase de la metodología de microingeniería.

Las habilidades señaladas en el modelo incremental de habilidades por fase están en orden de impacto y únicamente se encuentran las más relevantes según la aplicación de la técnica de Pareto (80-20). Cada una de las habilidades debe ser desarrollada y fortalecida a través de juegos, ejercicios prácticas. En el libro ya citado de Isauro Blanco [11] hay ejemplos de estas actividades. En el anexo D se muestran ejemplos de ejercicios a desarrollar para practicar las habilidades aplicando el modelo de construcción de algoritmos apoyado en heurísticas.

Capítulo 4. Experimentación de Campo

Con la intención de validar el marco teórico del Modelo de Construcción de Algoritmos Apoyado en Heurísticas se planteó desde un principio la comprobación experimental del modelo. Para ello, se ha realizado un experimento de campo utilizando estudiantes de primer cuatrimestre de Ingeniería de la Universidad del Valle de México Campus Hispano. En el presente capítulo se explica el desarrollo del mismo, su contexto y sus resultados.

El experimento de campo ha sido dividido en tres fases:

1. Planeación.- Conjunto de actividades que se desarrollaron antes de la aplicación del experimento y que tienen por objetivo asegurar su correcta ejecución.
2. Ejecución.- Conjunto de actividades que se realizan durante el experimento y tienen por objetivo su correcta aplicación.
3. Análisis de Resultados. Conjunto de actividades que se desarrollan después del último día de clases y tienen por objetivo recopilar, concentrar y analizar los datos obtenidos durante su ejecución.

El diseño del experimento, definición de variables, hipótesis, método y demás elementos formales están descritos con detalle en las secciones 1.2 a 1.5.

4.1 Planeación del Experimento de Campo

El éxito del experimento depende de su buen diseño y de su correcta aplicación. Con la intención de garantizar estos elementos se elaboró un plan que incluye estrategias y tácticas que aspiran a mantener el control de la ejecución del mismo y aportar rigor y confiabilidad en los resultados.

Considerando el método a seguir expresado en la sección 1.5, hay seis factores críticos que pueden alterar el éxito:

- Control de variables extrañas.
- El diseño y planeación didáctica del curso.
- La integración de los grupos de Control y Experimental.
- La selección de profesores.
- Selección y aplicación de instrumentos de preprueba y posprueba.
- Plan de introducción del Modelo de Construcción de Algoritmos.

Estos factores se analizan a detalle en las siguientes secciones.

4.1.1. Planeación del Control de las Variables Extrañas

Las variables extrañas son factores externos al estímulo que pueden afectar al resultado y ocasionar la pérdida de validez del experimento. Para disminuir su influencia en la variable dependiente se consideraron varias estrategias de control que a continuación son detalladas.

4.1.1.1. Control de Antecedentes

Al inicio del proceso, tanto el grupo experimental como el de control, a pesar de su heterogeneidad interna, deben ser homogéneos entre ellos. Para ello se debe demostrar la igualdad de antecedentes entre ambos grupos lo que permitirá que al evaluar las habilidades humana para diseñar y construir algoritmos imperativos (variable dependiente denominada Y), al finalizar el proceso, las diferencias estadísticas entre ellos puedan ser atribuidas al estímulo creado por el Modelo de Creación de Algoritmos Apoyado en Heurísticas (variable independiente denominada X).

Es decir, los grupos de 1er cuatrimestre de Ingeniería están formados por estudiantes aleatorios con antecedentes distintos entre sí. Aunque la gran mayoría no tiene estudios previos en programación, puede ser que algunos de ellos si los tengan, por lo tanto se aplican pruebas al inicio del curso, a modo de diagnóstico, para conocer el perfil del grupo. Si bien es claro que al interior de los grupos los estudiantes son heterogéneos (proviene de distintas preparatorias públicas o privadas, tienen distintos promedios, etc.) los grupos entre si deben tener igualdad en sus perfiles iniciales (denominado Yb).

La descripción de los Perfiles de Grupo al iniciar el experimento incluyen los siguientes factores:

- Número total de estudiantes.
- Distribución de la especialidad de la ingeniería que cursan sus miembros.
- Distribución de edades.
- Resultados de la prueba de Pronóstico de Éxito Académico (PEA)
- Resultados de pruebas de habilidades verbales y numéricas (College Board)
- Resultados de prueba de antecedentes en cultura informática.

En algunos factores aplican criterios de antecedentes para considerar homogéneos los grupos y son los siguientes:

- Número total de estudiante por grupo: Mínimo 20 máximo 25
- Distribución de edades: Mínimo 17 máximo 24 años
- Resultados de PEA: Diferencia máxima de 10% entre el grupo experimental y la media de resultados.
- Resultados de College Board: Que no haya diferencias estadísticas significativas.
- Resultados de la prueba de antecedentes informáticos: Diferencia mínima de .25 entre el grupo experimental y la media de resultados.

4.1.1.2. Control de Maduración

Al terminar el proceso se espera que todos los grupos hayan logrado mejoras en sus habilidades. Este es un criterio de validez del experimento ya que de no existir maduración tanto en el grupo experimental como en el de control significaría que hubo una falla en su aplicación y tendría que ser descartado. Para demostrar la maduración de los grupos se aplica una batería de pruebas que conforman el perfil final (denominado Ya). Este perfil incluye los siguientes factores

- Mortalidad experimental (bajas durante el curso) (denominado B)
- Resultados de evaluaciones finales del ciclo escolar

4.1.1.3. Control de Bajas

Otro factor que puede hacer perder la validez del experimento es la mortalidad experimental que corresponde al número de bajas que cada grupo tiene. Números elevados sesgarían los resultados de las pruebas de Ya y esto debe evitarse.

Si bien es normal que en grupos de 1er cuatrimestre se de el mayor índice de bajas de toda la carrera oscilando alrededor del 15% del total de su población, para este experimento los grupos deberán permanecer en un índice por debajo del 10%. Por otro lado tanto el grupo experimental (Ge) como el de control (Gc) serán comparados sus índices de bajas y entre ellos no podrá haber más de un 5% de diferencia entre ellos.

El control de bajas se llevará a cabo utilizando el sistema de Control Escolar de la UVM Campus Hispano y las listas de asistencia de los profesores involucrados en el experimento los cuales aportarán el índice de bajas oficial del ciclo para cada grupo.

4.1.1.4. Control de Instrumentación

La elección, consistencia y aplicación de los instrumentos que medirán Ya y Yb tanto de Ge como de Gc son factores críticos que son explicados con más detalle en la sección 4.1.5.

4.1.1.5. Control de Reactividad al Estimulo

Para disminuir la posible diferencia de reactividad al estímulo durante el proceso se han considerado las siguientes acciones:

- Igualdad en el Diseño del curso.- El temario, plan, prácticas de laboratorio, bibliografía, etc. son idénticos en los grupos (ver sección 4.1.2)
- Conformación de grupos homogéneos.- La distribución poblacional es aleatoria y deben cumplir los criterios citados en la sección 4.1.1.1. a través de pruebas se valida que sus perfiles de ingreso sean homogéneos (ver sección 4.1.3)

- Asignación de profesores con el mismo nivel de desempeño.- Se asignan profesores cuyos antecedentes sean equivalentes al igual que su desempeño docente (ver sección 4.1.4).
- Aplicación del mismo conjunto de instrumentos de medición.- Los instrumentos son los mismos y su forma de aplicación se hace en las mismas condiciones (ver sección 4.1.5).
- Asignación de mismo esquema de horario, aulas y laboratorios.- Tanto Ge como Gc deben tener el mismo horario de clase, laboratorios, aulas y calendario escolar (ver secciones 4.1.3 y 4.1.7).

4.1.2. Diseño y Planeación Didáctica del Curso

El diseño del curso es un documento que incluye los objetivos de la materia, los temas que se deben desarrollar, la bibliografía, el paquete de prácticas, los mecanismos de evaluación y una planeación detallada de los tópicos que deben ser abordados cada sesión de clase. Este documento es desarrollado por los cuerpos académicos en su conjunto por lo que la materia de Principios de Programación tiene un plan de ejecución que debe ser respetado por los profesores que imparten la materia ya que ellos en su conjunto lo elaboran. El diseño completo de esta materia puede consultarse en el anexo A-1.

El temario contempla la introducción a las computadoras y a la programación concentrándose en las estructuras de control de secuencia, condición e iteración y teniendo un alcance hasta arreglos. El lenguaje de programación que será utilizado es C# basado en la herramienta de Microsoft© Visual Estudio en su versión de 2003. Este temario deberá desarrollarse en 14 semanas con 3 horas de sesión a la semana haciendo un total de 42 horas totales.

Este documento fue consensuado por los cuatro profesores involucrados en su ejecución el jueves 1° de septiembre de 2005 al término de la junta previo inicio a clases incluyendo el detalle de los temas a abordar cada sesión.

4.1.3. Integración de Grupos de Control y Experimental

La apertura de los grupos será por libre demanda de los aspirantes. Se abrirán tantos grupos como estudiantes se matriculen. Cada grupo será de un máximo de 25 estudiantes y un mínimo de 20. Para garantizar su aleatoriedad, la conformación de estudiantes por grupo será por orden de inscripción.

El primer grupo que se integró será el experimental y asignado al profesor Ricardo Vargas de Basterra. Esta decisión es tomada en función de que si solo hubiese uno, en él se desarrollará el experimento cambiando su diseño de preprueba-estímulo-posprueba con grupo de control a preprueba-estímulo-posprueba eliminando el grupo de control. Si se registran más grupos estos serán asignados de acuerdo a los criterios dictados en la Selección de Profesores, sección 4.1.4, y serán considerados como grupos de control.

Como parte del plan se preparan los horarios hasta para cuatro grupos. Esta decisión es tomada en función de las metas de inscripciones que el Departamento de Mercadotecnia de la UVM Campus Hispano se ha fijado para Ingeniería.

Con la intención de que los horarios no sean factor diferenciador el 1er grupo de control y el experimental son asignados al mismo horario y de misma forma los siguientes dos, quedando como se muestra en la tabla 30:

No. de Grupo	Clave UVM asignada	Horario	Total de Horas a la semana	Ubicación de Aulas	Laboratorio Asignado
1*	02c	Martes de 9:00 a 12:00 horas	3	Campus B	B
2	04c	Martes de 9:00 a 12:00 horas	3	Campus B	B
3	06c	Jueves de 9:00 a 12:00 horas	3	Campus B	B
4	08c	Jueves de 9:00 a 12:00 horas	3	Campus B	B

Tabla 30. Integración de grupos

* Este es el grupo considerado Experimental.

4.1.4. Selección de Profesores

La selección de profesores para la ejecución del experimento es uno de los aspectos más delicados del mismo ya que ésta es una de las variables extrañas que más podrían afectar al resultado tanto a favor como en contra. Para asegurar la validez de este criterio se aplicaron las siguientes acciones:

4.1.4.1. Definición de Criterios

Para poder controlar este fenómeno y evitar impactos indeseados se consideraron tres criterios en la selección de los candidatos.

1. Evaluación de Desempeño.- Los profesores considerados deben estar en la misma categoría de evaluación para la materia de Principios de Programación de acuerdo al sistema de Evaluación de profesores de la UVM.
2. Experiencia.- Los elegidos deben tener experiencia de al menos cuatro ciclos lectivos impartiendo materias similares.
3. Conocimiento.- Los candidatos deben demostrar mismo nivel de conocimientos y habilidades de programación.

4.1.4.2. Aplicación de los Criterios

- En materia de Evaluación de Desempeño, la Universidad del Valle de México cuenta con un sistema de evaluación de profesores que contempla cuatro grandes aspectos:

1. Evaluación basada en la opinión de los estudiantes.- Consiste en la aplicación de un instrumento electrónico en el cual los estudiantes evalúan a sus profesores considerando

factores tales como la forma en que planean, ejecutan y evalúan el proceso de aprendizaje de los alumnos. Se incluye en el anexo B-1 copia del instrumento de evaluación.

2. Evaluación de pares.- Esta evaluación está basada en que el profesor evaluado es observado por uno o más docentes durante sus sesión y estos elaboran un reporte de sus observaciones utilizando un instrumento específicamente creado para este efecto.
3. Evaluación Administrativa.- Integrada por un conjunto de indicadores tales como índices de puntualidad y asistencia; entrega puntual de documentos y actas; asistencia y participación en academias; y participación en los programas de capacitación docente.

En el Anexo B-2 se incluye el resultado resumen de finales del ciclo lectivo 02/05 (mayo a agosto de 2005) con el cual se hizo la 1ª selección de candidatos.

- En materia de Experiencia se consideró específicamente la relacionada con materias de programación. La UVM en este momento se encuentra en etapa de introducción de nuevos planes de estudio por lo que la materia de Principios de Programación es la 1ª vez que se impartirá. Sin embargo, en los planes anteriores hay materias totalmente equivalentes en contenido pero con nombres distintos y que fueron consideradas como válidas para demostrar la experiencia requerida. Entre las materias que fueron contempladas para comprobar dicha experiencia se encuentran las siguientes: Lógica de programación, Introducción a la Programación, Programación Avanzada, Programación Orienta a Objetos, Diseño Algorítmico, Algoritmos y Estructuras de Datos.

- Para validar los conocimientos suficientes por parte de los profesores se utilizaron como instrumentos válidos los siguientes: Carrera profesional afín a la informática incluyendo; Cursos específicos de programación y/o de la herramienta a utilizar (en este caso C#); y certificaciones de empresas reconocidas en el área de programación. La tabla 31 muestra el cuadro resumen de perfiles de los profesores.

Nombre	Perfil	Antigüedad	Materias Impartidas	Evaluación	Observaciones
Díaz Bautista Ana Ma.	Especialidad en Comercio Electrónico Lic. en Informática administrativa	01/03/1995 10 años de experiencia en la institución	+Análisis y diseño de sistemas +Evaluación de sistemas de información +Graficación +Ingeniería de software +Diseño de Algoritmos +Lógica de Programación +Introducción a la Programación	C. Admón.24.80% P. Academias.25% S. Docente. 27.75% Cáp... actuli. 20% Total: 97.55% Nivel: 61.5 (B)	Certificación en Microsoft .net con avance hasta la tercera estrella. Curso certificación de C# de la ANIEI
Díaz Salinas José Luís	Especialidad en Comercio Electrónico	12/11/2002 3 años de experiencia	+Lógica de programación +Programación	C. Admón. 25% P. Academias. 24%	Certificación en Microsoft .net con avance

Nombre	Perfil	Antigüedad	Materias Impartidas	Evaluación	Observaciones
	Ing. en Sistemas computacionales	en la institución	orientada a objetos +Estructura de datos +Implantación, pruebas y mantenimiento de software +Programación Avanzada	S. Docente. 27.38% Cáp... actuli. 20% Total: 96.38% Nivel: 54.92 (B-)	hasta la tercera estrella. Curso certificación de C# de la ANIEI
Rodríguez López Felipe Armando	Especialidad en Comercio Electrónico Lic. en Informática administrativa	01/09/1996 9 años de experiencia en la institución	+Diseño de bases de datos +Estrategias de negocios +Fundamentos de computación +Sistemas operativos +Compiladores +Lógica de programación +Programación avanzada	C. Admón. 17.90% P. Academias. 17% S. Docente. 28.50% Cáp... actuli. 13.60% Total: 77.00% Nivel: 65.09 (B)	Certificación en Microsoft .net con avance hasta la primera estrella. Curso certificación de C# de la ANIEI
Vargas de Basterra Ricardo	Pasante Doctorado en ciencias computacionales Mtro. En Ciencias computacionales Ing. en Cibernética y ciencias de la computación	01/10/1996 9 años de experiencia en la institución	+Programación avanzada +Lógica de Programación + Diseño de Base de Datos + Ing. de Software + Proyectos de Ing. de Software	C. Admón. 25% P. Academias. 25% S. Docente. 13.88% Cáp... actuli. 13.10% Total: 76.98% Nivel: 54.1 (B-)	Curso certificación de C# de la ANIEI

Tabla 31. Cuadro resumen del perfil de profesores candidatos

4.1.4.3. Selección Propuesta

El primer candidato a profesor será Ricardo Vargas de Basterra quien tendrá bajo su responsabilidad el grupo experimental. Esta decisión es tomada en base a su conocimiento sobre el modelo de construcción de algoritmos apoyado en heurísticas.

El segundo candidato será Felipe Armando Rodríguez López quien tendrá bajo su responsabilidad el grupo de control. Esta decisión es tomada en base a que es el profesor cuyo perfil es el más semejante al del profesor con el grupo experimental.

El tercer candidato será Ana María Díaz Bautista quien tendrá al tercer grupo que se apertura de esta materia. Esta decisión es tomada en base a las políticas de la UVM que dicta que los grupos deben de ser asignados en orden iniciando en los mejor evaluados. En este caso Ana María Díaz es la mejor evaluada.

El cuarto candidato será José Luís Díaz Salinas quien tendrá el siguiente grupo que se apertura de esta materia. Esta decisión es tomada en base a que es el segundo profesor mejor evaluado.

4.1.5. Selección y Aplicación de Instrumentos de Preprueba y Posprueba

Los instrumentos a utilizar tanto para medir el perfil de ingreso (Yb), como el de egreso (Ya) (a través de la prueba de habilidades de diseño y construcción de algoritmos), son uno de los factores críticos y que podrían ocasionar la anulación del experimento. De los resultados que se obtengan de estos instrumentos se aceptará o rechazará Hi. Es por ello que se ha decidido aplicar una serie de normas que permitan mantener la confiabilidad del estudio. Estas son las siguientes:

- Aplicar la misma batería de pruebas para Ya en todos los grupos
- Aplicar la misma batería de pruebas para Yb en todos los grupos
- Utilizar las mismas condiciones de aplicación

4.1.5.1. Batería de Pruebas para Determinar el Perfil de Ingreso (Yb)

El perfil de ingreso representa el estado en que el grupo inicia el proceso. Como resultado de la aplicación de esta batería de pruebas se debe de contar con una visión sobre las habilidades de los grupos antes de iniciar el proceso.

Las pruebas seleccionadas son las siguientes:

- Prueba de Perfil de Éxito Académico (PEA).- Es un complejo instrumento que se aplica institucionalmente a todos los alumnos de primer ingreso a la UVM y evalúa la probabilidad de éxito académico de los estudiantes.
- Prueba de College Board.- La prueba de College Board es un instrumento internacional que evalúa las habilidades verbales y las habilidades numéricas de las personas. La UVM lo aplica institucionalmente a todos sus alumnos de primer ingreso para conocer el perfil de ingreso.
- Prueba de antecedentes de Cultura Informática.- Este es un cuestionario a través del cual se evalúan los conocimientos que tienen los estudiantes en materia de informática y computación. Particularmente se evalúa el conocimiento sobre sistemas operativos, ofimática, Internet y programación.

4.1.5.2. Batería de Pruebas para Determinar el Perfil de Egreso (Ya)

El perfil de egreso representa el estado en que el grupo termina el proceso. Al aplicar esta batería de pruebas se debe de disponer de la visión sobre las habilidades desarrolladas por los grupos al terminar el proceso.

La batería de pruebas a aplicar consiste en la prueba de Habilidades de Diseño y Construcción de Algoritmos denominada HaDiCA misma que se explica con detalle en la sección 4.1.5.4.

4.1.5.3. Condiciones de Aplicación

Para conservar la validez del experimento también es importante que las pruebas se hayan aplicado en igualdad de circunstancias para todos los grupos. Los criterios aplicados son los siguientes:

- Misma duración.- Todos los grupos deben de tener el mismo tiempo para contestar los instrumentos.
- Misma ubicación.- Todos los grupos deben de realizar las pruebas en instalaciones con las mismas características de amplitud, mobiliario, iluminación, ventilación y confort.
- Misma facilidades.- Todos los grupos deben de disfrutar de las mismas facilidades tales pizarrón, audiovisuales, servicios sanitarios, servicio de estacionamiento, servicio médico, etc.
- Misma fechas.- Todas las pruebas para Yb deben hacerse en el transcurso de la 1er semana de clase y las de Ya en la última.

En caso de faltar a alguna de estas condiciones la prueba se anulará y en consecuencia el experimento.

La diferencia entre Ya y Yb es el índice de crecimiento (madurez alcanzada) que desarrollaron los grupos en el proceso del experimento. Sin embargo, no es el criterio para aceptar o rechazar la hipótesis, esto depende, como se explicó, de las diferencias entre los Ya entre el grupo experimental y el de control.

4.1.5.4. Prueba de Habilidades de Diseño y Construcción de Algoritmos (HaDiCA)

Esta prueba es el componente de mayor peso en Ya porque debe evaluar específicamente las habilidades de diseño y construcción de algoritmos. Debido a que es un factor crítico en la validez del experimento, y con la intención de cumplir con el rigor que se requiere, se establecieron un conjunto de especificaciones que debe de cumplir para que pueda ser considerada adecuada para el caso de estudio.

4.1.5.4.1. Especificaciones de HaDiCA

Hay cuatro rasgos prioritarios que esta prueba de habilidades debe de contemplar:

- Independencia
- Centrada en habilidades
- Claridad en la medición
- Adaptativa

Por *Independencia* se entiende que los resultados obtenidos de la aplicación del instrumento no sean afectados por elementos ajenos o complementarios a las habilidades. Es decir, los resultados que arroje la prueba deben ser inmunes a circunstancias tales como si un estudiante trabajó con C

y otro con Pascal; uno con programación orientada a objetos y otro en estructurada; uno en Unix y otro en Windows; etc. Se han distinguido cuatro tipos de independencia que la prueba tiene que cumplir:

- Debe ser independiente de la teoría de programación
- Debe ser independiente de la metodología utilizada
- Debe ser independiente del lenguaje
- Debe ser independiente de cualquier herramienta de programación

Centrada en Habilidades significa que lo que se mide es la capacidad humana de resolver problemas, en este caso por medio del análisis, diseño y construcción de algoritmos. No debe ser la intención de la prueba medir los conocimientos teóricos almacenados en la memoria de los estudiantes sino su capacidad de utilizarlos para resolver problemas prácticos. Son cuatro las habilidades principales que se requieren en la construcción de algoritmos:

- Analizar un problema real.
- Diseñar una solución algorítmica.
- Construir un programa basado en el algoritmo.
- Probar el programa a la luz del problema original

Claridad de Medición implica que la prueba debe ser lo más objetiva posible y con resultados de fácil interpretación, no solo numéricos sino también reflexivos. El software en si mismo es un intangible y cada problema podría tener varias soluciones algorítmicas distintas y la prueba deberá ser capaz de dar valores justos a cada solución. Así mismo, los resultados obtenidos deben ser de fácil interpretación y con capacidad para retroalimentar al evaluado. Las características de medición que el instrumento debe cumplir son las siguientes:

- Objetividad
- Cualitativo-cuantitativo
- Perfil de resultados de fácil interpretación y lectura
- Capacidad para la retroalimentación

La *Adaptabilidad* consiste en que los reactivos que conforman la prueba puedan ajustarse a las circunstancias de aplicación en los grupos conservando su validez y escala de medición. Los requisitos para que la adaptabilidad sea correcta son:

- Equivalencia de complejidad.
- Inalterabilidad de la métrica.

4.1.5.4.2. Diseño de HaDiCA

Considerando las particularidades que esta prueba debe cumplir y tras revisar en la ACM y en la IEEE se tomó la decisión de hacer un diseño específico que satisfaga todas las condiciones antes expuestas.

Para el diseño del instrumento se analizaron distintos tipos de reactivos y se tomó la decisión de diseñar el instrumento utilizando las características de integración de escenarios de la técnica denominada Aprendizaje Basado en Problemas (PBL por sus siglas en Inglés) y para la evaluación y métrica de resultados la técnica denominada Rúbrica [85]. Combinando estos conceptos, la estructura de HaDiCA está formada por los siguientes elementos:

- Hoja de problemas que incluye escenarios de PBL
- Grupo de rúbricas analíticas.
- Propuesta de solución para cada escenario de PBL

El aprendizaje basado en problemas es toda una metodología que incluye una currícula y un proceso de aprendizaje, según describen sus autores Barrows y Tamblyn [7]. Sin embargo, lo único que se tomará de ella es la técnica para integrar escenarios de problemas ya que estos tienen ventajas específicas para el caso de estudio que se desea hacer.

Los escenarios de PBL tienen las siguientes características [88]:

- Se explican en términos de un problema de la vida real
- No sugieren como resolverlo
- No desarrolla ni evalúa conocimientos sino capacidades de solución de problemas
- Genera múltiples hipótesis
- Requiere esfuerzo para resolverlo
- Es consistente con los resultados de aprendizaje deseados
- Se construye sobre conocimientos y experiencias previas
- Promueve el desarrollo de habilidades mentales superiores

Gracias a que estas características coinciden con las especificaciones de diseño de la prueba de habilidades se utilizaron sus lineamientos para el diseño de los reactivos.

Una rúbrica o matriz de valoración es un instrumento utilizado para medir el desempeño de los estudiantes en áreas donde los mecanismos de medición tradicionales no reflejan el logro del aprendizaje. Se cita textual de [37]:

“Una Matriz de Valoración (Rúbrica - Rubric en inglés) facilita la Calificación del desempeño del estudiante en las áreas del currículo (materias o temas) que son complejas, imprecisas y subjetivas. Esta Matriz podría explicarse como un listado del conjunto de criterios específicos y fundamentales que permiten valorar el aprendizaje, los conocimientos y/o las competencias, logrados por el estudiante en un trabajo o materia particular.”

Hay dos tipos de rúbricas, las comprensivas u holísticas y las analíticas. Las rúbricas holísticas se usan para evaluar más la totalidad del proceso o producto. La rúbrica analítica o específica se utiliza cuando se evalúan diferentes aspectos de un concepto. En este caso se usará la rúbrica analítica

La rúbrica está compuesta por un conjunto de criterios y una escala de medición con descriptores claramente definidos, en este caso corresponde a las habilidades que se medirán en la

prueba. Cada habilidad, en función de sus descriptores, recibirá un puntaje ajustado a una escala Likert. Se eligieron escalas pares para evitar la tendencia central y obtener resultados más objetivos. La acumulación de estos puntos brindará un perfil detallado por categoría de habilidad y la suma total corresponde al nivel alcanzado de habilidad en diseño y construcción de algoritmos. En el anexo C se puede ver el diseño completo de HaDiCA incluyendo sus tres componentes en forma detallada.

En la tabla 32 se pueden observar las categorías de habilidades y los puntajes máximos y mínimos que pueden obtener por problema resuelto (cada problema es un escenario).

Habilidad	Descriptor	Mínimo	Máximo
Análisis	Resultados	1	4
	Procesos	1	4
	Entradas necesarias	1	4
	Restricciones y/o condiciones	1	4
	Iteraciones o repeticiones	1	4
	Diccionario de datos	1	4
Total de la Habilidad		6	24
Diseño	Secuencias	1	4
	Condiciones	1	4
	Iteraciones	1	4
	Pruebas	1	4
	Integración	1	4
	Consistencia	1	4
Total de la Habilidad		6	24
Codificación	Funcionalidad	1	4
	Confiabilidad	1	4
	Usabilidad	1	4
	Eficiencia	1	4
	Mantenibilidad	1	4
	Portabilidad	-	-
	Consistencia	1	4
Total de la Habilidad		6	24
Total de cada escenario		18	72
Gran total de la prueba (2 escenarios)		36	144

Tabla 32. Puntaje de HaDiCA

El puntaje máximo que puede obtener una persona que aplica la prueba es de 144 puntos totales (suponiendo que enfrenta dos escenarios de PBL) y el mínimo es de 36.

4.1.5.4.3. Aplicación, Calificación e Interpretación de HaDiCA

Como consecuencia del diseño de la prueba se desprenden las siguientes condiciones de aplicación.

- Se deben resolver al menos dos de cuatro escenarios expuestos.
- La duración total de la prueba es de máximo tres horas.

- Se pueden utilizar recursos bibliográficos de apoyo.
- El uso de equipo de cómputo es altamente recomendado. Si no estuviera disponible, entonces ningún grupo deberá de hacer uso de computadoras (por principio de igualdad de aplicación).
- Debe ser realizada en forma individual.

Para calificar la prueba se utiliza la rúbrica analítica diseñada para cada escenario de la siguiente forma:

- Cada escenario debe haber sido completamente contestado por su diseñador.
- Se aplican las métricas mostradas en el anexo C del diseño de HaDiCa.
- En función de las métricas se definen los rangos para la escala de Likert de cada descriptor de cada habilidad.
- Se revisan las respuestas de la persona evaluada indicando la escala correspondiente de cada respuesta de cada descriptor de cada habilidad.
- Se asigna el puntaje de cada descriptor y se obtiene el total de cada descriptor sumando los valores obtenidos en cada escenario resultado.
- Una vez totalizados los puntos obtenidos por la suma de los descriptores de cada escenario se vuelven a sumar para obtener el total por cada habilidad.
- La suma de puntos de todas las habilidades de cada escenario contestado determina su calificación final o índice de habilidad.

Es importante señalar que la calificación final es un índice acumulado para una valoración cuantitativa, sin embargo, de acuerdo a las especificaciones explicadas en la sección 4.1.5.4.1 lo que se busca es un perfil analítico que permita la fácil lectura y retroalimentación al evaluado.

Para interpretar los resultados de la prueba se debe elaborar el perfil a partir de los totales de cada descriptor. Estos se convierten en ejes en un diagrama de los denominados “radial” o “telaraña”. Es posible integrar un perfil acumulado por habilidad (ver figura 77) o descomponerlo en sus descriptores (ver figura 78) para poder hacer una interpretación más completa.

A modo de ejemplo se presentan datos ficticios y su interpretación en la tabla 33.

HABILIDAD	DESCRIPTOR	MÍN.	MÁX	ESC. 1	ESC. 2	TOTAL
Análisis	Resultados	2	8	4	3	7
	Procesos	2	8	3	3	6
	Entradas necesarias	2	8	3	3	6
	Restricciones y/o condiciones	2	8	3	2	5
	Iteraciones o repeticiones	2	8	1	2	3
	Diccionario de datos	2	8	2	3	5
Total de la Habilidad		12	48	16	16	32
Diseño	Secuencias	2	8	4	4	8
	Condiciones	2	8	3	4	7
	Iteraciones	2	8	3	2	5
	Pruebas	2	8	2	2	4
	Integración	2	8	3	3	6
	Consistencia	2	8	2	2	4

HABILIDAD	DESCRIPTOR	MÍN.	MÁX	ESC. 1	ESC. 2	TOTAL
Total de la Habilidad		12	48	17	17	34
Codificación	Funcionalidad	2	8	3	4	7
	Confiabilidad	2	8	3	3	6
	Usabilidad	2	8	2	3	5
	Eficiencia	2	8	3	3	6
	Mantenibilidad	2	8	2	2	4
	Consistencia	2	8	3	3	6
Total de la Habilidad		12	48	16	18	34
GRAN TOTAL DE LA PRUEBA (2 escenarios)		36	144	49	51	100

Tabla 33. Ejemplo de puntaje

En el ejemplo, se muestran los resultados que obtuvo un estudiante al resolver dos problemas denominados Escenario 1 y Escenario 2 así como sus totales respectivos. El Índice de Habilidad es de 100 puntos que están conformados por un subíndice de 32 puntos para análisis, 34 para diseño y 34 para construcción. Cada uno de los subíndices de habilidades está integrado por los valores de sus descriptores. En el caso del análisis está formado por 7 puntos de resultados, 6 de procesos, 6 de entradas, 5 de restricciones, 3 de iteraciones y 5 de diccionario de datos. Estos valores son graficados de tal manera que se puede tener una fácil lectura de su perfil global o analítico.

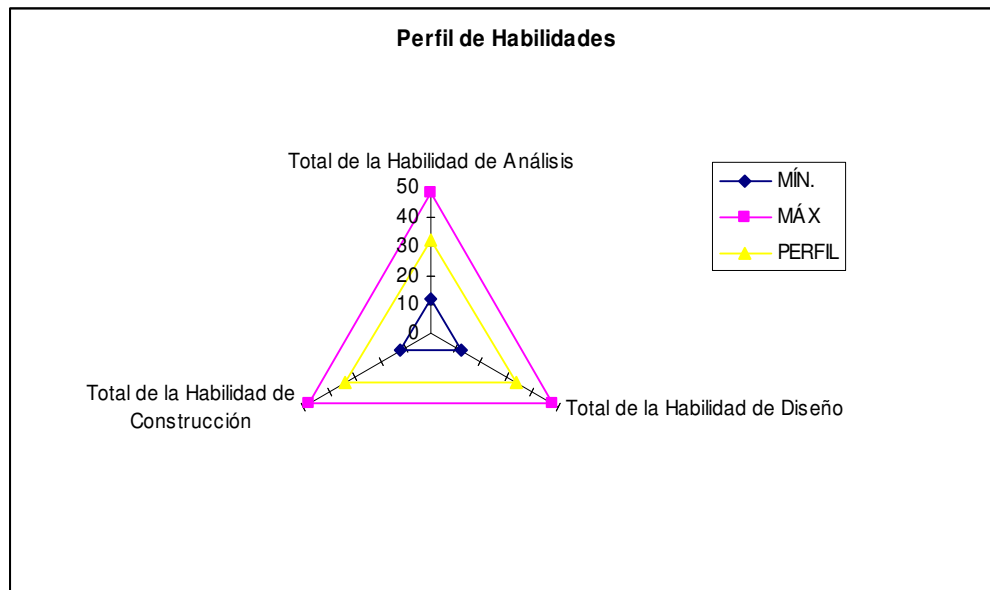


Figura 77. Perfil por habilidades

A simple vista se puede observar cuales son las habilidades en que hay fortaleza y cuales los que requieren trabajo para su mejoría. En el ejemplo se observa que el análisis de iteraciones es el elemento más débil del proceso mientras que el diseño de secuencias es la más fuerte.

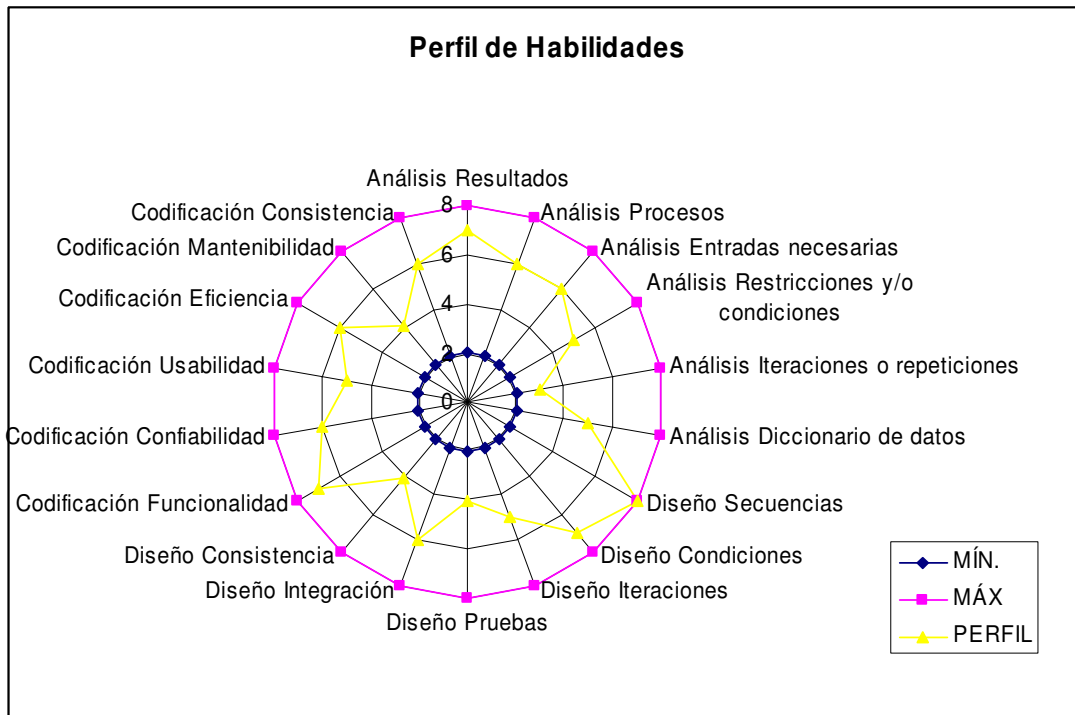


Figura 78. Perfil por descriptores

4.1.6. Plan de Introducción del Modelo de Construcción de Algoritmos Apoyados en Heurísticas

Debido a que el actual contenido de la materia Principios de Programación está registrado ante la Secretaría de Educación Pública con clave 5B2709 no pueden eliminarse temas de su contenido pero si pueden agregarse nuevos elementos para su enriquecimiento. Por lo tanto, se partió del diseño original de la planeación didáctica que fue impartida por todos los profesores y puede verse en el anexo A-1. A esta planeación original se le agregaron los elementos del modelo diseñando los temas, actividades tanto de enseñanza como de aprendizaje, el orden y el calendario de sesión en que se abordarían. El resultado final se observa en el anexo A-2 el cual contiene el plan de introducción explicado con detalle y utilizando los formatos institucionales de la Universidad.

Los grupos de control 04c, 06c y 08c utilizarán la planeación original mientras que el grupo experimental 02c utilizará la versión modificada de la materia.

4.1.7. Calendario

Partiendo del principio de que el experimento se desarrollará en un ciclo de clases normales de un cuatrimestre es necesario planear el calendario escolar y las principales actividades que se desarrollarán dentro de él. El experimento durará los cuatro meses completos de acuerdo al siguiente calendario mostrado en la tabla 34.

Fechas	Actividad
29 de Agosto al 2 de Septiembre	Inscripciones y entrega de horarios del ciclo 03/05.
1 de Septiembre.	Junta de profesores previo inicio a clases.
5 de Septiembre.	Inicio del ciclo escolar.
15 y 16 de Septiembre.	No laborales.
3 al 7 de Octubre.	1er. Periodo de evaluaciones.
1 y 2 de Noviembre.	No laborales.
7 al 11 de Noviembre.	2º Periodo de evaluaciones
21 de Noviembre	Inicio de evaluación de profesores
12 al 15 de Diciembre	3er. Periodo de evaluaciones
13 al 16 de Diciembre	Entrega de resultados finales
14 de Diciembre	Fin de clases
19 al 30 de Diciembre	Periodo Vacacional
2 al 6 de Enero	Reinscripciones
9 de Enero	Reinicio de Clases ciclo 01/06

Tabla 34. Calendario de actividades

4.2 Ejecución del Experimento de Campo

Durante esta etapa se ejecutaron las actividades planificadas y explicadas en la sección 4.1 cuyo objetivo es asegurar el rigor metodológico y el éxito del experimento. Debido a esto, las actividades están en estricta correspondencia con dicha planeación.

El proceso de ejecución se dividió en dos grandes vertientes de acciones:

- Acciones propias de control de las Variables Extrañas para mantener el proceso dentro de los parámetros previamente establecidos.
- Acciones propias de la planeación didáctica tanto del grupo experimental como de los grupos de control.

4.2.1. Acciones de Control de Variables Extrañas

Debido a la naturaleza experimental de campo de esta investigación el control de las variables extrañas es fundamental para la validez de las conclusiones. La descripción de las mismas y el plan de control se explican en la sección 4.1.1 y los resultados de su ejecución se abordan a continuación.

4.2.1.1. Integración de Grupos

El criterio de validez señala que los grupos deben ser de conformación aleatorizada.

El experimento inició con la integración de los grupos, el cual se dio a partir del proceso de inscripción que se desarrolló del 29 de agosto al 2 de septiembre en tiempo y forma. Al concluir el proceso ya se tenía una población de 89 estudiantes totales inscritos en la materia de Principios de la Programación para el ciclo 03/05 iniciando clases el 5 de septiembre de 2005. Esta población se distribuyó en cuatro grupos cuyas claves oficiales y matrícula se muestran en la tabla 35.

Clave de Grupo	Estudiantes	Profesor Asignado
02c	24	Ricardo Vargas de Basterra
04c	25	Felipe Armando Rodríguez López
06c	22	Ana María Díaz Bautista
08c	18	José Luís Díaz Salinas

Tabla 35. Grupos integrados con profesor

La distribución de los alumnos en los grupos fue totalmente al azar sin intervención extraña. El procedimiento fue estrictamente en orden de solicitud. Los responsables de la integración de los grupos fue el área de Atención al Público en combinación con Control Escolar.

El grupo 02c fue designado como grupo experimental y los otros tres como posibles grupos de control. La validez del grupo experimental y los de control depende de que cumplan con los criterios definidos en la pruebas de diagnóstico inicial.

La asignación de profesores fue apegado a los criterios fijado en la sección 4.1.4, asignando el primero grupo al prof. Ricardo Vargas de Basterra, el segundo a Felipe Armando Rodríguez López, el tercero a Ana María Díaz, y el cuarto a José Luís Díaz Salinas.

4.2.1.2. Control de Antecedentes

El criterio de validez señala que los grupos deben ser homogéneos en su perfil de arranque. Esto se logra demostrando los siguientes puntos explicados en la sección 4.1.1.1:

- Número total de estudiante por grupo: Mínimo 20 máximo 25
- Distribución de edades: Mínimo 17 máximo 24 años
- Resultados de PEA: Diferencia máxima de 10% ente el grupo experimental y la media de resultados.
- Resultados de College Board: Que no haya diferencias estadísticas significativas entre las muestras de habilidades verbales y de habilidades numéricas.
- Resultados de la prueba de antecedentes informáticos: Diferencia mínima de .25 entre el grupo experimental y la media de resultados.

El grupo 08c no cumple con el criterio de número total de estudiantes por lo que se descarta del experimento.

4.2.1.2.1. Número Total de Estudiantes

La tabla 36 contiene el número total de estudiantes de cada grupo

CONCEPTO	GRUPOS			
	02c	04c	02c	04c
Número total de Estudiantes	22	23	24	18
Cumple	Si	Si	Si	No

Tabla 36. Población en los grupos candidatos

Como se puede observar únicamente el grupo 04c no cumple con el criterio por lo que se le descarta del experimento.

4.2.1.2.2. Distribución de Edades

La tabla 37 contiene el análisis de estadística descriptiva de las edades de los grupos

Análisis de edades				
Concepto	02c	04c	06c	08c
Media	18.5	18.52	18.875	19.11111111
Error típico	0.31851103	0.26532998	0.38688884	0.52324333
Mediana	18	18	18	18.5
Moda	18	18	18	18
Desviación estándar	1.560379	1.32664992	1.89536047	2.21993346
Varianza de la muestra	2.43478261	1.76	3.5923913	4.92810458
Curtosis	0.51301186	0.18415379	8.25706071	4.59740623
Coefficiente de asimetría	1.23596346	0.76793033	2.57764837	1.83919533
Rango	5	5	9	9
Mínimo	17	17	17	17
Máximo	22	22	26	26
Suma	444	463	453	344
Cuenta	24	25	24	18
Nivel de confianza (95.0%)	0.65889026	0.54761417	0.80034053	1.10394693

Tabla 37. Análisis estadístico de edades en la población

El criterio marca que las edades debes ser entre 17 y 24 años, por lo tanto los grupos 06c y 08c son descartados porque ambos tienen estudiantes con edad mayor al límite.

4.2.1.2.3. Resultados de la Prueba de Pronóstico de Éxito Académico

El criterio señala que la diferencia máxima entre el grupo experimental y la media de resultados sea de 10%. En la tabla 38 se muestran los resultados de la evaluación

Concepto	02c		04c		06c		08c		Media	Diferencia
3 @	6	25.00%	11	44.00%	7	29.17%	4	22.22%	30.10%	-5.10%
2 @	14	58.33%	9	36.00%	13	54.17%	11	61.11%	52.40%	5.93%
1 @	3	12.50%	3	12.00%	4	16.67%	1	5.56%	11.68%	0.82%
Sin Evaluar	1	4.17%	2	8.00%	0	0.00%	2	11.11%	5.82%	-1.65%
Totales	24	100.00%	25	100.00%	24	100.00%	18	100.00%		

Tabla 38. Resultados de evaluación PEA

Como se puede observar la diferencia más alta es de 5.93% en el caso de 2@ por lo que todos los grupos cumplen el criterio.

4.2.1.2.4. Resultado de la Prueba de College Board

Durante las primeras semanas de actividad se aplicaron las pruebas de habilidades verbales y habilidades numéricas del College Board. Para demostrar la homogeneidad de aplica la prueba estadística *t* de *student* para muestras pequeñas con un valor de confianza alpha de 0.05. La hipótesis es que ambas muestras no tienen diferencias estadísticas significativas y por lo tanto son homogéneas.

En la tabla 39 se presentan los resultados de estadística descriptiva en las muestras de todos los grupos.

Análisis de Aptitud Verbal				
Concepto	02c	04c	06c	08c
Media	522.409091	480.590909	487.904762	493.066667
Error típico	19.1771937	16.5944312	22.10795	17.0875802
Mediana	533.5	487.5	510	510
Moda	584	446	538	547
Desviación estándar	89.9490115	77.8347817	101.311354	66.1799135
Varianza de la muestra	8090.82468	6058.25325	10263.9905	4379.78095
Curtosis	0.1246273	-0.40031393	0.33790966	-0.12771279
Coeficiente de asimetría	0.13680331	-0.19814951	-0.6899163	-0.80364437
Rango	368	296	398	222
Mínimo	362	325	233	362
Máximo	730	621	631	584
Suma	11493	10573	10246	7396
Cuenta	22	22	21	15
Nivel de confianza (95.0%)	39.8811574	34.5100088	46.1163754	36.6492144

Tabla 39. Análisis estadístico de la prueba de aptitudes verbales

En la tabla 40 se muestran los resultados del cálculo de la prueba *t*. entre el grupo 02c (experimental) y el 04c (de control) que hasta el momento ha cumplido con todos los criterios de aceptación. Se puede observar que el valor crítico para *t* es de 2.0195 y el valor de *t* es de 1.6489. Como *t* es menor que su valor crítico se acepta la hipótesis de que no hay diferencias estadísticas significativas y por lo tanto son grupos homogéneos en habilidad verbal

Prueba t para dos muestras suponiendo varianzas desiguales		
alpha = 0.05	Habilidad Verbal	
	02c	04c
Media	522.409091	480.590909
Varianza	8090.82468	6058.25325
Observaciones	22	22
Diferencia hipotética de las medias	0	
Grados de libertad	41	
Estadístico t	1.64896872	
P(T<=t) una cola	0.05339833	
Valor crítico de t (una cola)	1.682878	
P(T<=t) dos colas	0.10679666	
Valor crítico de t (dos colas)	2.01954095	

Tabla 40. Resultados de prueba *t* de student de aptitudes verbales

La tabla 41 contiene los valores de estadística descriptiva de las distribuciones de la prueba de habilidad numérica para todos los grupos .La habilidad numérica es considerada importante para poder desarrollar posteriormente la competencia de la programación.

Análisis de Aptitud Numérica				
Concepto	02c	04c	06c	08c
Media	607.285714	563.47619	548.5	550.266667
Error típico	17.3322082	22.7507843	26.6901815	27.2748736
Mediana	613	553	553	553
Moda	684	392	533	644
Desviación estándar	79.4261562	104.257191	119.36212	105.635131
Varianza de la muestra	6308.51429	10869.5619	14247.3158	11158.781
Curtosis	-0.89494427	-0.62490229	-0.22755007	-1.3465854
Coefficiente de asimetría	-0.42245384	0.09146488	-0.10826099	-0.25325702
Rango	276	383	457	313
Mínimo	452	392	341	381
Máximo	728	775	798	694
Suma	12753	11833	10970	8254
Cuenta	21	21	20	15
Nivel de confianza (95.0%)	36.1543527	47.4573043	55.8631918	58.4987856

Tabla 41. Análisis estadístico de la prueba de habilidades numéricas

En la tabla 42 se muestran los resultados del cálculo de la prueba *t* entre los grupos 02c y 04c. Se puede observar que el valor crítico de *t* es de 2.0210 y *t* tiene un valor de 1.4536. Como *t* es menor que su valor crítico se considera que no hay diferencia estadística significativa entre ambos muestras y por lo tanto son homogéneos en la habilidad numérica.

Prueba t para dos muestras suponiendo varianzas desiguales		
alpha = 0.05	Habilidad numérica	
	02c	04c
Media	601.136364	560.227273
Varianza	6840.02814	10584.184
Observaciones	22	22
Diferencia hipotética de las medias	0	
Grados de libertad	40	
Estadístico t	1.4536324	
P(T<=t) una cola	0.07692486	
Valor crítico de t (una cola)	1.68385101	
P(T<=t) dos colas	0.15384972	
Valor crítico de t (dos colas)	2.02107537	

Tabla 42. Resultados de prueba t de student de aptitudes numéricas

4.2.1.2.5. Resultado de la Prueba de Antecedentes en Cultura Informática

Las figuras 79 a 82 muestran los perfiles de inicio del curso en materia de conocimientos previos de informática y computación basados en la Prueba de Antecedentes de Cultura Informática.

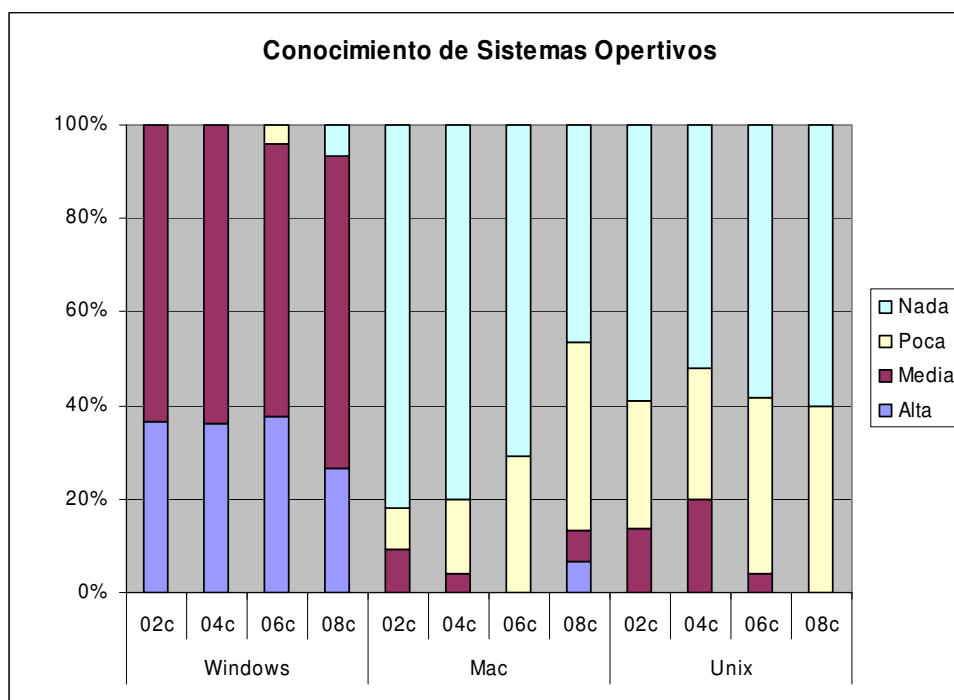


Figura 79. Perfiles de sistemas operativos

El perfil de sistemas operativos evalúa los conocimientos que tienen los sustentantes sobre tres distintos sistemas operativos. En la figura 79 se puede observar, además de la igualdad del perfil,

que existe un amplio dominio de conocimiento de Windows contra sus contrapartes de Mac o cualquier versión de Unix (incluyendo Linux).

La figura 80 muestra los perfiles de conocimientos en ofimática en dónde se observa que la herramienta con mayor dominio (en los cuatro grupos) es Word, seguido de Power Point, Excel y finalmente Access.

La figura 81 muestra los perfiles de conocimientos en Internet en dónde se puede observar que la Web y el correo electrónico tienen un alto nivel de conocimiento, no así el caso del diseño de páginas con HTML

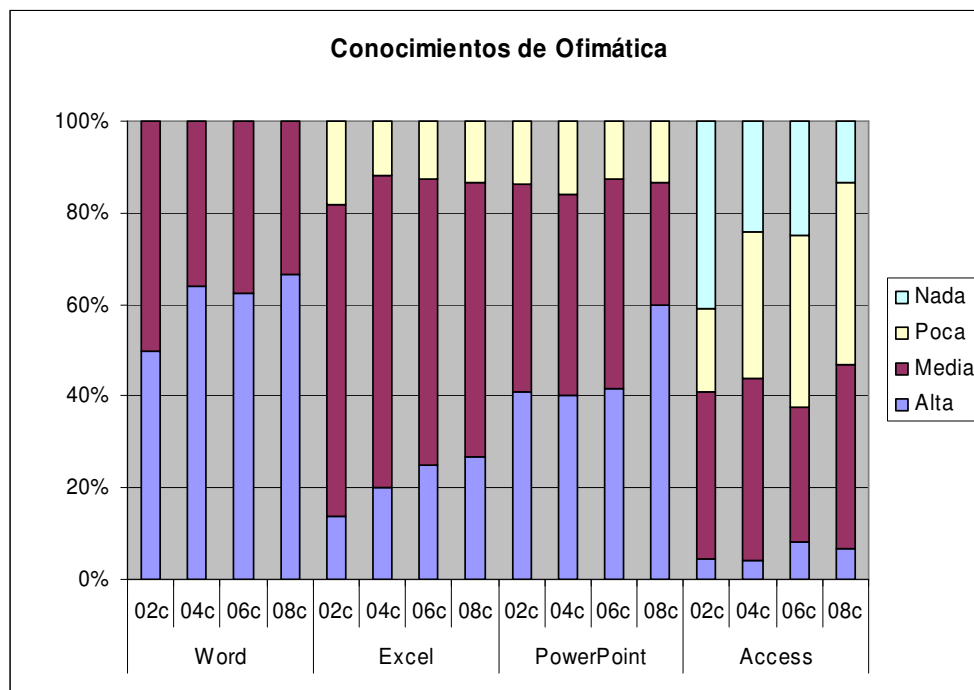


Figura 80. Perfiles en ofimática

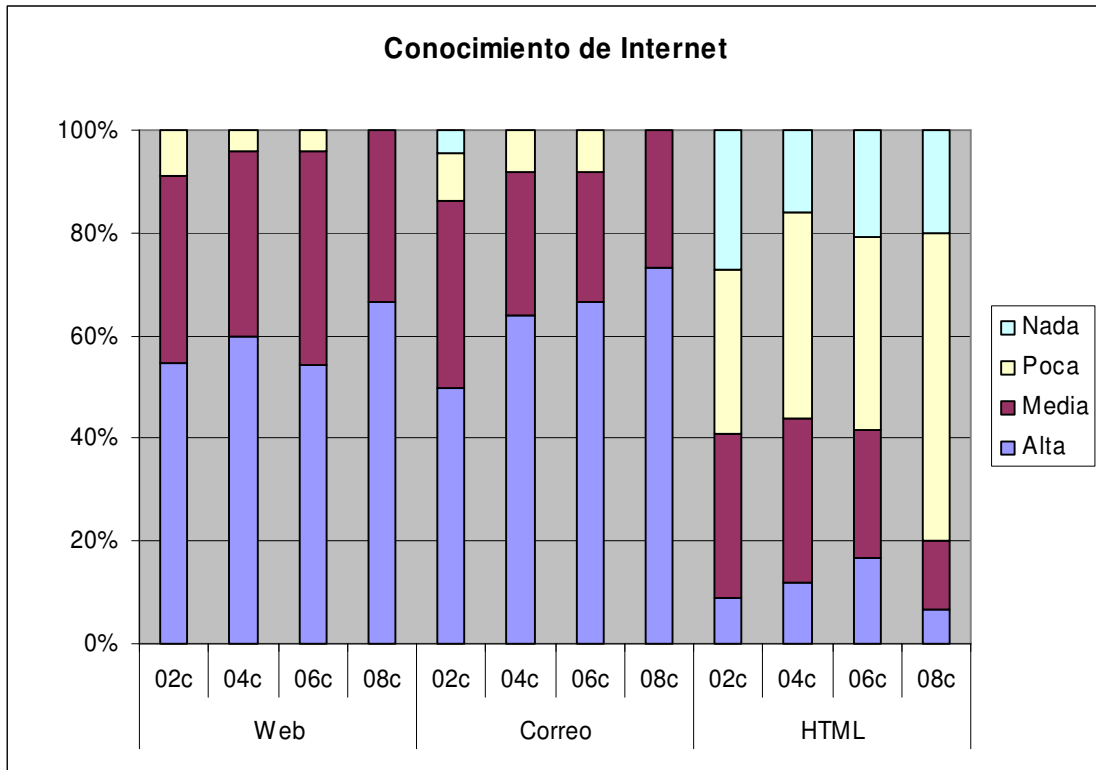


Figura 81. Perfiles en Internet

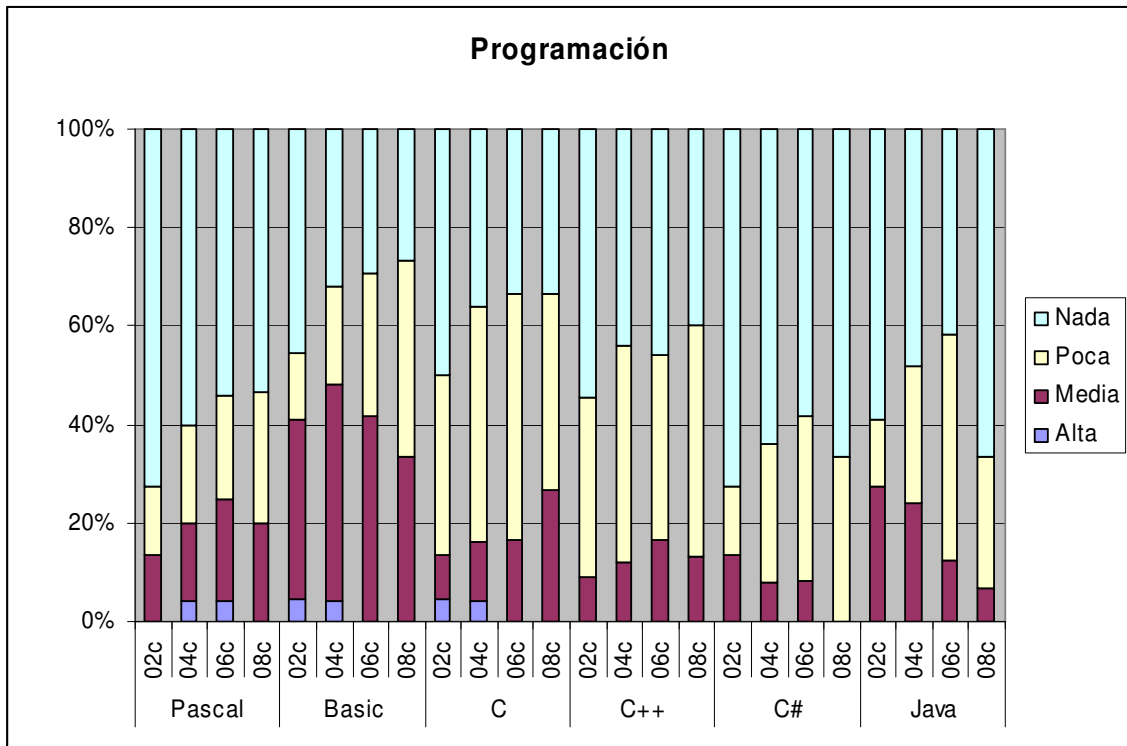


Figura 82. Perfiles en programación

La figura 82 muestra los perfiles en conocimientos de programación. En ella se aprecia que hay alguna ligera experiencia previa en Basic pero que se mantiene igual en los cuatro grupos. Siguiendo a Basic hay también un muy ligero despunte de Pascal, C y Java. El resto de los lenguajes tienen bajos índices de conocimiento

La tabla 43 contiene el promedio de evaluación por cada rubro de la prueba de antecedentes para cada uno de los grupos y de los cuatro perfiles de las gráficas anteriores.

Tema	02c	04c	06c	08c	Promedio	Dif. con la media
Sis. Operat.	4.12	4.19	4.06	4.18	4.14	-0.01
Ofimática	5.89	6.12	6.13	6.33	6.12	-0.23
Internet	6.86	7.12	7.08	7.40	7.12	-0.25
Programación	3.42	3.70	3.71	3.49	3.58	-0.16

Tabla 43. Evaluación de antecedentes técnicos

El criterio señala que para aceptar la homogeneidad el valor de cada perfil no puede ser de más de 0.25 de diferencia con la media lo cual se cumple en todos los casos.

4.2.1.3. Control de Maduración

El criterio de validez señala que todos los grupos deben tener mejorías en sus habilidades

En la tabla 44 se muestra un análisis de las calificaciones finales de cada uno de los cuatro grupos. Si se comparan con los valores iniciales resultado de su prueba de antecedentes de cultura informática en el rubro de programación se observará que hay una diferencia sustantiva en el avance que han logrado todos los grupos a lo largo de su ciclo escolar.

Análisis de Promedios Finales				
Concepto	02c	04c	06c	08c
Calificación promedio	7.39393939	7.9047619	8.20833333	7.47916667
Error típico	0.40891718	0.17986305	0.23468595	0.16092542
Mediana	7.66666667	7.66666667	8.33333333	7.33333333
Moda	7.66666667	7.66666667	9	7
Desviación estándar	1.91799157	0.82423606	1.14972166	0.64370168
Varianza de la muestra	3.67869168	0.67936508	1.3218599	0.41435185
Curtosis	1.78964047	-0.48424516	2.05409594	-0.99884441
Coeficiente de asimetría	-1.32568449	0.73582424	-1.18584108	0.36228012
Rango	7.33333333	2.66666667	5	2
Mínimo	2.66666667	7	5	6.66666667
Máximo	10	9.66666667	10	8.66666667
Suma	162.666667	166	197	119.666667
Cuenta	22	21	24	16
Nivel de confianza (95.0%)	0.85038982	0.37518775	0.48548488	0.34300441

Tabla 44. Análisis de promedios finales

Tema	02c	04c	06c	08c	Promedio	Dif. con la media
Sistemas Operativos	4.12	4.19	4.06	4.18	4.14	-0.01
Ofimática	5.89	6.12	6.13	6.33	6.12	-0.23
Internet	6.86	7.12	7.08	7.40	7.12	-0.25
Programación	3.42	3.70	3.71	3.49	3.58	-0.16

Tabla 45. Resultados en prueba de antecedentes aplicada al inicio del ciclo

En las tablas 45 y 46 se muestran los resultados de la prueba de antecedentes aplicados al inicio del ciclo escolar y el análisis comparativo de maduración utilizando una prueba aplicada al final del ciclo escolar.

Grupo	Evaluación inicial	Evaluación Final	Diferencia
02c	3.42	7.39	+3.97
04c	3.7	7.9	+4.20
06c	3.71	8.2	+4.49
08c	3.49	7.47	+3.98
Promedio	3.58	7.74	+4.16

Tabla 46. Análisis comparativo de maduración

4.2.1.4. Control de Bajas

El criterio señala que para que un grupo sea válido debe tener un índice de bajas de 10% o menos y la diferencia entre cada grupo no puede ser mayor a 5%.

Concepto	02c	04c	06c	08c
Matricula inicial	22	23	24	18
Bajas	2	1	1	2
Porcentaje de bajas	9.09%	4.35%	4.17%	11.11%
Matricula final	20	22	23	16
Número de aprobados	20	22	22	16
Número de no aprobados	2	0	1	0
Índice de reprobación	10.00%	0.00%	4.55%	0.00%
Índice de aprobación	90.00%	100.00%	95.45%	100.00%

Tabla 47. Control de bajas

En la tabla 47 se aprecia que el grupo 08c tuvo 2 bajas al igual que el grupo 02c, pero debido a su población este número de bajas rebasa el 10% de su población por lo que este grupo es descartado y no califica para las pruebas de aceptación o rechazo de la hipótesis.

Los grupos 02c, 04c y 06c cumplen con los dos criterios por lo que son considerados validos para el experimento.

4.2.1.5. Control de Instrumentación

El criterio señala que las prepruebas y pospruebas deben ser las mismas y aplicarse bajo las mismas condiciones.

Las prepruebas consisten en tres instrumentos:

- Prueba de habilidades verbales y numéricas del College Board
- Prueba de pronóstico de éxito académico (PEA)
- Encuesta de antecedentes computacionales

Las pruebas de College Board se aplican contra reloj contando únicamente con 20 minutos para cada habilidad. Este criterio fue respetado en los cuatro grupos.

La prueba de pronóstico de éxito académico tiene una duración máxima de 30 minutos y el criterio fue respetado para los cuatro grupos.

La encuesta de antecedentes tiene una duración de 15 minutos y el criterio fue respetado en todos los grupos.

Tanto College Board como PEA fueron aplicadas por personal del departamento de Desarrollo de Estudiante cumpliendo con rigor sus criterios ya que incluso la información es procesada por empresas externas a la UVM quienes entregan un reporte integrado de resultados. La encuesta de antecedentes fue aplicada por los propios profesores de cada materia. En todos los casos fueron los mismos instrumentos y condiciones de aplicación.

La posprueba consiste en la aplicación de la Prueba de habilidades de diseño y construcción de algoritmos (HaDiCA)

La prueba HaDiCA debe ser aplicada con una duración de hasta 3 horas pudiendo utilizar equipo de cómputo para su aplicación. Los grupos 02c, 04c y 06c cumplieron con este criterio pero el grupo 08c únicamente les dio una hora para responder la prueba lo que invalida sus resultados por no cumplir el criterio de igualdad de condiciones.

En todos los casos fueron las mismas pruebas pero únicamente en tres de los grupos se aplicaron bajo las mismas condiciones.

4.2.1.6. Control de Reactividad al Estímulo

Esta sección agrupa varios criterios definidos en la sección 4.1.1.5 que a continuación se resumen:

- Igualdad en el diseño del Curso. Tal como se explica en la sección 4.1.2, los cuatro profesores utilizaron el mismo temario, la misma planeación didáctica, el mismo paquete de prácticas y la misma bibliografía. Únicamente el grupo experimental agregó a su

planeación didáctica los temas del Modelo de Construcción de Algoritmos Apoyado en Heurísticas. Con estas condiciones se concluye que se cumplió el criterio.

- Conformación de grupos homogéneos. De acuerdo al análisis presentado en la sección 4.2.1.2 los grupos son homogéneos con lo que se cumple el criterio.
- Asignación de profesores con el mismo nivel de desempeño. Este criterio es desarrollado en la sección 4.1.4 en el que se demuestra que los profesores tienen el mismo perfil y nivel de desempeño por lo que cumplen con este criterio.
- Aplicación del mismo conjunto de instrumentos de medición. La sección 4.2.1.5 analiza esta situación y concluye con que cumplió el criterio en tres de los grupos, 02c, 04c, 06c; descartando el 08c ya que no cumplió con el criterio.
- Asignación del mismo esquema de horarios, aulas y laboratorios. La sección 4.1.3 describe como los cuatro grupos fueron asignados a aulas de las mismas condiciones, se asignó el mismo laboratorio y se definió exactamente el mismo esquema de horarios de tres horas por sesión y una sesión a la semana. Con lo que se cumple con el criterio.

4.2.2. Acciones de la Planeación Didáctica

Las clases iniciaron en tiempo y forma de acuerdo a calendario. Los profesores, aulas, horarios y espacios de laboratorio fueron asignados y aplicados de acuerdo a la planeación académica del curso cumpliendo a su vez con las especificaciones del experimento.

Para la tercera semana de haber iniciado el curso ya se había terminado de aplicar las pruebas diagnósticas de PEA y de College Board por parte del departamento de Desarrollo de Estudiante del Campus y la encuesta de antecedentes por parte de los profesores. Los resultados se explican en la sección 4.2.1.2.

Los cuatro grupos utilizaron el mismo laboratorio de cómputo y, en base al acuerdo previo entre los profesores, se aplicó el mismo paquete de prácticas mostrado en el anexo A-3 aunque hubo variación en el calendario de entrega para cada grupo pero todos cumplieron.

Los cuatro grupos utilizaron el mismo paquete de planeación didáctica estandarizado en la UVM (descrito en el anexo A-1) y únicamente el grupo experimental (grupo 02c) agregó a su planeación didáctica los temas del modelo de construcción de algoritmos apoyado en heurísticas conservando todos los temas ya existentes (ver el anexo A-2). El material adicional en este grupo ocasionó que durante las primeras cinco sesiones se retrasara con respecto a los otros grupos. Siendo los grupos 04c y 06c (impartidos por los profesores Felipe Armando Rodríguez y Ana María Díaz respectivamente) los más avanzados seguidos por una clase del 08c (impartido por José Luís Díaz Salinas) y el más retrasado era el grupo experimental 02c (del profesor Ricardo Vargas). Para enfrentar esta situación varios de los temas de la planeación didáctica básica se dejaron como temas de investigación individual y en clase solo se discutieron dudas dejando ejercicios de prácticas para complementarlos. A pesar de estas acciones el grupo experimental siempre se mantuvo dos clases atrás de los punteros pero pudo mantener el ritmo de forma que concluyó solamente a una clase de diferencia igualando al grupo 08c que también se había atrasado una clase.

Durante el período del experimento los grupos 06c y 08c tuvieron su horario a la misma hora todos los jueves por lo que realizaron actividades los días 8, 22 y 29 de septiembre, 6, 13, 20 y 27 de octubre, 3, 10, 17 y 24 de noviembre y el 1 y 8 de diciembre teniendo un total de 13 sesiones completas faltándoles una sesión para completar las 14 planeadas. Los grupos 02c y 04c tuvieron su horario a la misma hora todos los martes por lo que realizaron sus actividades los días 6, 13, 20 y 27 de septiembre, 4, 11, 18 y 25 de octubre, 8, 15, 22 y 29 de noviembre y el 6 de diciembre teniendo un total de 13 sesiones faltándoles también una para completar el plan inicial. Esto ocasionó que los cuatro grupos vieran parcialmente el último tema que correspondía a arreglos.

Durante las dos primeras semanas de noviembre se realizó la evaluación de profesores correspondiente a la opinión de los alumnos y durante la última de noviembre y primera de diciembre se realizaron las otras etapas correspondientes a la evaluación académica y la administrativa. Los resultados se encuentran en el anexo B-3.

Los periodos de evaluación fueron los mismos para los cuatro grupos de acuerdo al calendario pero las técnicas y exámenes no lo fueron por lo que las calificaciones no pueden ser utilizadas para la aceptación o rechazo de la hipótesis ya que no cumplen con el criterio de estandarización de instrumentos.

Durante diciembre se aplicaron las pospruebas de habilidades verbales y numéricas por parte del departamento de apoyo al estudiante. El día 6 los grupos 02c y 04c presentaron la prueba de Habilidades de Diseño y Construcción de Algoritmos (HaDiCA) y el 8 de diciembre la aplicó el grupo 06c. Estos tres grupos tuvieron igualdad de tiempo y recursos para responder la prueba, sin embargo, el grupo 08c solamente tuvo una hora para su aplicación por lo que se desechó su validez y quedó fuera del experimento al violar el criterio de igualdad de condiciones de aplicación del instrumento.

Durante el cuatrimestre completo los profesores Felipe Armando Rodríguez López (grupo 04c) y Ana María Díaz Bautista (grupo 06c) no tuvieron una sola falta. Los profesores José Luís Díaz Salinas (grupo 08c) y Ricardo Vargas de Bastera (grupo 02c) tuvieron una falta cada uno pero el último hizo una clase de reposición para poder completar un conjunto de trece sesiones completas.

En resumen, los grupos 02c (experimental), 04c y 06c (grupos de control) cumplieron con todos los requisitos establecidos como condiciones para considerarse válido el experimento. El grupo 08c fue descartado ya que no cumplió con la estandarización de aplicación de las pruebas y tuvo una clase menos que el resto.

4.2.3. Conclusiones de la Ejecución del Experimento

Al concluir el experimento se habían recabado todos los datos necesarios para poder aceptar o rechazar la hipótesis. Este análisis se presenta en la sección 5.2.

A modo de resumen se presenta la tabla 48 en la que se observan los criterios de rigor que los grupos de control y experimental debían de cumplir y si el criterio fue satisfecho para poder utilizar los datos obtenidos en el proceso de análisis.

criterio	02c*	04c	06c	08c
Sobre el perfil inicial de los grupos				
Los grupos se conforman de manera aleatoria	Si	Si	Si	Si
La población de los grupos debe ser de un máximo de 25 y un mínimo de 20 estudiantes	Si	Si	Si	No
La distribución de edades debe ser equivalente	Si	Si	No	No
Los resultados de la prueba PEA deben ser equivalentes	Si	Si	Si	Si
Los resultados de la prueba College Board deben ser equivalentes	Si	Si	Si	Si
Los resultados del cuestionario de antecedentes en materia de conocimientos de sistemas operativos deben ser equivalentes	Si	Si	Si	Si
Los resultados del cuestionario de antecedentes en materia de conocimientos de herramientas de ofimática deben ser equivalentes	Si	Si	Si	Si
Los resultados del cuestionario de antecedentes en materia de conocimientos de Internet deben ser equivalentes	Si	Si	Si	Si
Los resultados del cuestionario de antecedentes en materia de conocimientos de Programación en lenguajes 3GL deben ser equivalentes	Si	Si	Si	Si
Sobre el perfil de los profesores				
Los profesores deben pertenecer a la misma categoría del sistema de evaluación de profesores de la Universidad previo al inicio de clases y como criterio de selección.	Si	Si	Si	Si
Los profesores deben tener al menos experiencia de cuatro ciclos lectivos impartiendo materias equivalentes	Si	Si	Si	Si
Los profesores deben demostrar mismo nivel de conocimientos en el área de programación de C#	Si	Si	Si	Si
Los profesores deben ser evaluados en la misma categoría por los cuatro grupos del curso al finalizar el curso.	Si	Si	Si	Si
Sobre el perfil final de los grupos				
El número total de estudiantes debe estar entre 20 y 25	Si	Si	Si	No
El porcentaje de bajas debe ser menor a 10%	Si	Si	Si	Si
La diferencia del índice de bajas entre los grupos no debe ser mayor a 5%	Si	Si	Si	No
Los resultados de las pruebas de habilidades después del curso deben ser equivalentes entre los grupos	Si	Si	Si	Si
Sobre la reactividad al estímulo y ejercicio didáctico				
El temario a desarrollar por los grupos debe ser igual	Si	Si	Si	Si
El paquete de prácticas debe ser el mismo para todos los grupos	Si	Si	Si	Si
La estructura de sus horarios debe ser igual	Si	Si	Si	Si
Los escenarios que utilizan (aulas y laboratorios) deben ser iguales	Si	Si	Si	Si
La bibliografía y otros elementos de apoyo deben ser los mismos	Si	Si	Si	Si
El número de sesiones de clase y avances deben ser los mismos	Si	Si	Si	No
Sobre el control de instrumentos				
Los instrumentos de preprueba deben ser los mismos	Si	Si	Si	Si
Las condiciones de aplicación de los instrumentos de preprueba deben ser iguales	Si	Si	Si	Si
Los instrumentos de posprueba deben ser los mismos	Si	Si	Si	Si
Las condiciones de aplicación de los instrumentos de posprueba deben ser iguales	Si	Si	Si	No
Conclusión				
Se Cumplió con todos los criterios y por consecuencia se	Si	Si	No	No

Criterio	02c*	04c	06c	08c
aceptan sus resultados como validos para su análisis estadístico y aceptación o rechazo de la hipótesis				

Tabla 48. Resumen de control de variables extrañas

* Grupo experimental

En conclusión, los grupos 02 y 04c cumplieron con todos los criterios. Los datos obtenidos de estos grupos se utilizarán en la etapa de análisis. Los grupos 06c y 08c no cumplieron con todos los criterios por lo que son descartados y los datos obtenidos de ellos no serán tomados en cuenta en el análisis.

Para efectos de análisis posteriores se consideran los grupos 02c y 04c, el primero como experimental y el segundo como de control.

Capítulo 5. Conclusiones

A lo largo del documento se fundamentan diversas conclusiones y reflexiones. En las secciones siguientes se presenta la conclusión del Modelo de Construcción de Algoritmos Apoyado en Heurísticas, el análisis de resultados del experimento de campo (incluyendo la prueba de hipótesis), una descripción de los objetivos alcanzados, resumen de aportaciones, trabajos futuros y palabras finales.

5.1 Conclusiones del Modelo CAAH

En la tabla 49 se presenta la integración del Modelo de Construcción de Algoritmos Apoyado en Heurísticas. Como ya explicó, este modelo cuenta con tres dominios (modelo teórico, modelo humano y modelo herramental) soportado por un conjunto de heurísticas que orientan sobre el diseño de los algoritmos imperativos, por ello la tabla cuenta con columnas descriptivas para su descripción. La tabla 26 en la sección 3.6.4 amplía y detalla la información del modelo.

Fase	Etapas	Producto	Herramienta	Habilidad [†]
Análisis	Resultados	Tabla de resultados [†] .	Método de Análisis [†] (basado en procedimientos “hacia atrás”).	CRS, CRM, ERS, ERM, CCS, CUM, CSM, MUF, NTS
	Procesos	Listado de procesos [†] .	Árbol de descomposición de procesos [†] . Método de Análisis [†] (basado en procedimientos “hacia atrás”).	NRS, NRM, DSS, DSM, NCS, DCS, ERS, ERM, ESS, ESM, NSS, CIS, MCS, ETS, ETM, NTS, NTM, CIS, MCS, NCS, DUS, MUS, MRS, ERS, NCS
	Datos de Entrada	Tabla de datos de entrada [†] .	Método de Análisis [†] (basado en procedimientos “hacia atrás”).	CUM, CRM, ERS, ERM, CCS
	Restricciones	Relación de restricciones [†] .	Tablas de decisiones. Árboles de Decisiones. Definiciones de dominio. Método de Análisis [†] (basado en procedimientos “hacia atrás”).	ERS, ERM, ESS, ESM, NSS, CIS, CRS, CRM, ERM, EIM, MUS, MRS, MCS, NCS, NRS, NRM, NIS, NIM, DCS, CIM, DIS, DIM, DCS, EIS
	Repeticiones	Relación de repeticiones [†] .	Definiciones de dominio y rangos. Método de Análisis [†] (basado en procedimientos “hacia atrás”).	CRS, CRM, ERS, ERM, EIM, CSS, CSM, ESS, ESM,
	Diccionario de Datos	Tabla de Diccionario de Datos.	Método de Análisis [†] (basado en	CCS, CCM, CUS, CUM, ECM, ERM,

Fase	Etapa	Producto	Herramienta	Habilidad [†]
			procedimientos “hacia atrás”).	ECS, ERS, CIS, ESS, ESM
Diseño	Secuencias	Diagramas de secuencia.	Método de Tránsito de Análisis a Diseño [†] . Productos del Análisis. LeMVI [†] . Heurísticas de diseño de algoritmos imperativos [†] . Herramienta de software creada bajo la arquitectura DAAC [†] .	NSS, NSM, NTS, NTM, DIM, DRS, MIS, NCS, DCS
	Condicionales	Diagramas condicionales.	Método de Tránsito de Análisis a Diseño [†] . Productos del Análisis. LeMVI [†] . Heurísticas de diseño de algoritmos imperativos [†] . Herramienta de software creada bajo la arquitectura DAAC [†] .	NSS, NSM, NTS, NTM, DIM, DRS, MIS, DCS
	Ciclos	Diagramas de ciclos cerrados y abiertos.	Método de Tránsito de Análisis a Diseño [†] . Productos del Análisis. LeMVI [†] . Heurísticas de diseño de algoritmos imperativos [†] . Herramienta de software creada bajo la arquitectura DAAC [†] .	NSS, NSM, NTS, NTM, DIM, DRS, MIS, DCS
	Pruebas	Tablas de casos de pruebas de caja negra. Tablas de casos de pruebas de caja blanca.	Método de Tránsito de Análisis a Diseño [†] . Productos del Análisis.	NSM, NIM, DSM, DIM
	Integración y refinamiento	Diagramas de tareas y procesos.	Método de Tránsito de Análisis a Diseño [†] . LeMVI [†] . Productos del Análisis. Heurísticas de diseño de algoritmos imperativos [†] . Herramienta de software creada bajo la arquitectura DAAC [†] .	NTS, NTM, NIS, NIM, NCS, DTS, DTM, DIS, DIM, DRS, MIS, NCS
Codificación	Trascripción a 3GL	Código en algún lenguaje de programación tipo 3GL.	Método de esqueletos [†] . Herramienta de software creada bajo la arquitectura DAAC [†] . Productos del Diseño.	ETS, ETM, NTS, NTM, CTS, MTS, DUS
	Compilación	Programa ejecutable.	Código en algún lenguaje 3GL.	ETS, ETM, NTS, NTM, CTS, MTS,

Fase	Etapas	Producto	Herramienta	Habilidad †
			Herramienta de software creada bajo la arquitectura DAAC [†] . Compilador del lenguaje en alguna herramienta tipo IDE.	DUS
Pruebas	Detección de errores	Relación de pruebas aplicadas y resultados obtenidos. Relación de errores, vicios u otros problemas.	Método de pruebas secuencia-condición-repetición [†] . Casos de Prueba de caja negra y de caja blanca. Productos del Diseño Productos de la codificación.	ESS, ESM, ERS, ERM, EIM, EIS, NIS, NIM, DIS, DIM
	Depuración y corrección	Código en algún lenguaje de programación tipo 3GL depurado. Programa ejecutable depurado.	Productos del Diseño Productos de la codificación. Heurísticas de diseño de algoritmos imperativos [†] . LeMVI [†] Herramienta de software creada bajo la arquitectura DAAC [†] .	ESS, ESM, ERS, ERM, EIM, EIS, NIS, NIM, DIS, DIM
Liberación	Integración documental	Paquete documental.	Relación de pruebas aplicadas y resultados obtenidos. Relación de errores, vicios u otros problemas. Documentos producidos en todas las fases previas.	MRS, MRM, MSS, MSM
	Control de versiones	Tabla de control de versiones y cambios.	Documentos producidos en todas las fases previas.	MRS, MRM, MSS, MSM
	Entrega	Producto de software y documentación.	No aplica.	MRS, MRM, MSS, MSM

Tabla 49. Integración del modelo CAAH

† Aportaciones originales de esta tesis.

El Modelo CAAH utiliza muchos componentes creados a lo largo del tiempo por distintos investigadores de las ciencias de la computación, pero también utiliza muchos componentes nuevos creados específicamente para esta investigación. En la tabla 49 se presenta el ensamble de todos esos productos, herramientas y habilidades. Se ha marcado con un símbolo de cruz (†) las aportaciones específicas de este trabajo.

No debe omitirse que, además de los elementos indicados como aportación específica en la tabla 39, los métodos, procedimientos y metodologías, tanto específicas para etapas o fases como integrales del modelo mismo también son aportaciones puntuales de este trabajo.

5.2 Análisis de Resultados del Experimento de Campo

En las secciones siguientes se presentan los resultados obtenidos de los grupos experimental y de control tanto en la parte previa al experimento como en su fase final. Este análisis permite soportar una de las conclusiones más importantes de este trabajo: la aceptación o rechazo de la hipótesis nula, cuyo criterio, según se definió en su momento, depende de si hay o no diferencia estadística significativa entre el grupo experimental y el de control.

5.2.1. Análisis de Resultados de Yb (al inicio del experimento)

Las pruebas aplicadas denominadas Yb tienen por objetivo demostrar que los grupos tanto experimental como de control son homogéneos al inicio del experimento. Esto quedó controlado y demostrado en el capítulo 4 donde se explica con detalle el análisis de cada criterio. Más información puede encontrarse en la sección 4.2.1.2 en la cual se muestran las tablas de datos y las gráficas correspondientes a cada uno de los grupos con su correspondiente evaluación para cumplir con el criterio de aceptación.

Como resultado de la aplicación de estas pruebas se definieron los grupos 02c como experimental y 04c como de control ya que solamente estos cumplieron con todas las condiciones de aplicación. Los otros dos grupos fueron desechados.

A continuación se presentan las tablas de índices iniciales a modo de resumen.

5.2.1.1. Índices Iniciales de PEA

En la tabla 50 se muestra el resumen de estudiantes que obtienen 3, 2 o 1 @ en el índice de PEA. Cada @ representa el grado de probabilidad de éxito académico.

Concepto	02c	04c	06c	08c
3 @	6	11	7	4
2 @	14	9	13	11
1 @	3	3	4	1
Sin Evaluar	1	2	0	2
Totales	24	25	24	18

Tabla 50. Índices de la prueba PEA

5.2.1.2. Índices Iniciales de College Board

En la tabla 51 se muestran los resultados obtenido en la Prueba College Board tanto en habilidad verbal como numérica de los grupos Experimental (02c) y de Control (04c)

Prueba de College Board Inicial			
Habilidad Verbal		Habilidad Numérica	
02c	04c	02c	04c
575	446	674	553
584	399	684	432
372	427	623	392
418	427	684	674
631	483	654	523
584	594	583	613
547	520	533	533
538	584	613	654
538	483	654	472
455	501	593	644
362	492	523	775
547	492	684	623
427	372	482	492
483	325	452	472
529	557	543	543
510	455	492	573
730	538	694	674
584	621	684	603
640	446	728	704
529	520	593	492
483	538	583	392
427	353	472	492

Tabla 51. Índices de la prueba College Board

5.2.1.3. Índices Iniciales de Prueba de Antecedentes Informáticos

La tabla 52 contiene el promedio de evaluación por cada rubro de la prueba de antecedentes para cada uno de los grupos.

Tema	02c	04c	06c	08c	Promedio	Dif. con la media
Sis. Operat.	4.12	4.19	4.06	4.18	4.14	-0.01
Ofimática	5.89	6.12	6.13	6.33	6.12	-0.23
Internet	6.86	7.12	7.08	7.40	7.12	-0.25
Programación	3.42	3.70	3.71	3.49	3.58	-0.16

Tabla 52. Promedio de evaluación de antecedentes técnicos

El grupo experimental (02c) inició el experimento estando por debajo del grupo de control (04c) en todos los rubros. Sin embargo, esto no es factor para descalificar la homogeneidad la cual se mide obteniendo la diferencia que hay entre la media de los grupos y el grupo experimental y este índice no puede ser mayor a un cuarto de punto (0.25) lo cual se cumple en todos los casos.

5.2.2. Análisis de Resultados de Ya (prueba de habilidades al final de proceso)

Las pruebas denominadas Ya tienen dos objetivos: El primero es demostrar que tanto el grupo de control como el experimental han generado madurez a través del ciclo escolar, lo cual se demuestra comparando Yb y Yb de tanto de Ge como de Gc; el segundo objetivo, y de mayor trascendencia, es comparar el Ya de Ge y de Gc para buscar diferencias estadísticas significativas.

La variable Ya fue medida básicamente a través de la aplicación de la Prueba de Habilidades de Análisis, Diseño y Codificación de Algoritmos (HADiCA). Información detallada sobre su diseño puede consultarse en el capítulo 4 sección 4.1.5.4.

Los resultados de HADiCA se resumen en las siguientes tablas de concentrados de datos y sus respectivas gráficas.

5.2.2.1. Resultados del Grupo Experimental (02C)

El grupo experimental obtuvo un índice global en la prueba de HaDiCA de 94.05 el cual se detalla en la tabla 53.

Grupo:02c

total: 94.05

Análisis	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
	7.50	6.65	6.40	5.60	3.10	5.90
Diseño	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
	6.30	5.10	3.65	2.10	5.20	5.45
Codificación	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
	5.70	5.30	5.25	4.45	5.25	5.15
Totales	Análisis	35.15	Diseño	27.80	Codifica	31.10

Tabla 53. Índice HaDiCA grupo experimental

Cómo puede observarse el resultado más alto se obtuvo en la fase de análisis con 35.15 puntos, seguido de la fase de codificación con 31.10. La fase con menor puntaje es la de diseño que se vio fuertemente impactada por una pobre habilidad en el diseño de pruebas que la arrastró hacia abajo logrando apenas 27.80 puntos siendo ésta el área más débil de todas. Por otra parte el área más fuerte es la de identificación de resultados lo cual es un paso crucial en la metodología en todas las metodologías de solución de problemas.

Se implementaron una serie de ajustes al modelo teórico inicial y el presentado actualmente en el capítulo tres ya tiene los ajustes para mejorar los resultados en las áreas de oportunidad.

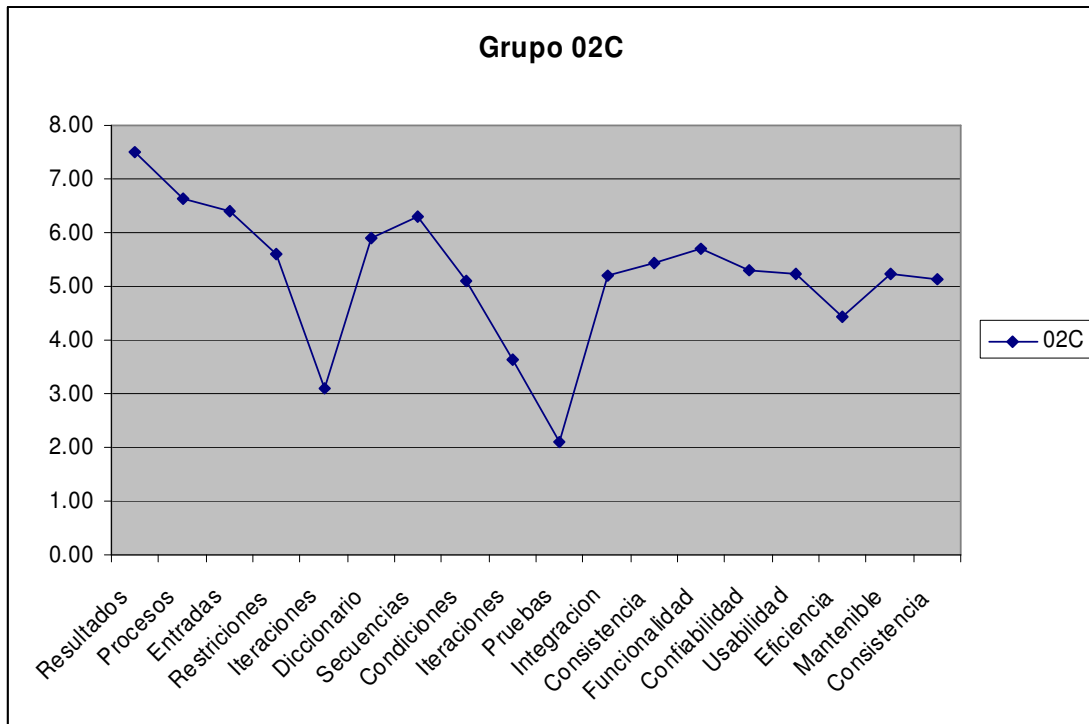


Figura 83. Histograma de habilidades específicas grupo experimental

En la Figura 83 se muestra el histograma con los valores que se obtienen por cada una de las áreas evaluadas por HaDiCA. Se observa que el valor más débil es el diseño de pruebas seguido por la identificación de iteraciones y luego por el diseño de las iteraciones. Se puede observar la clara correlación que existe entre una baja identificación de las iteraciones que impacta también en un bajo puntaje en el diseño de las mismas. El resto de los valores se presentan en niveles aceptables.

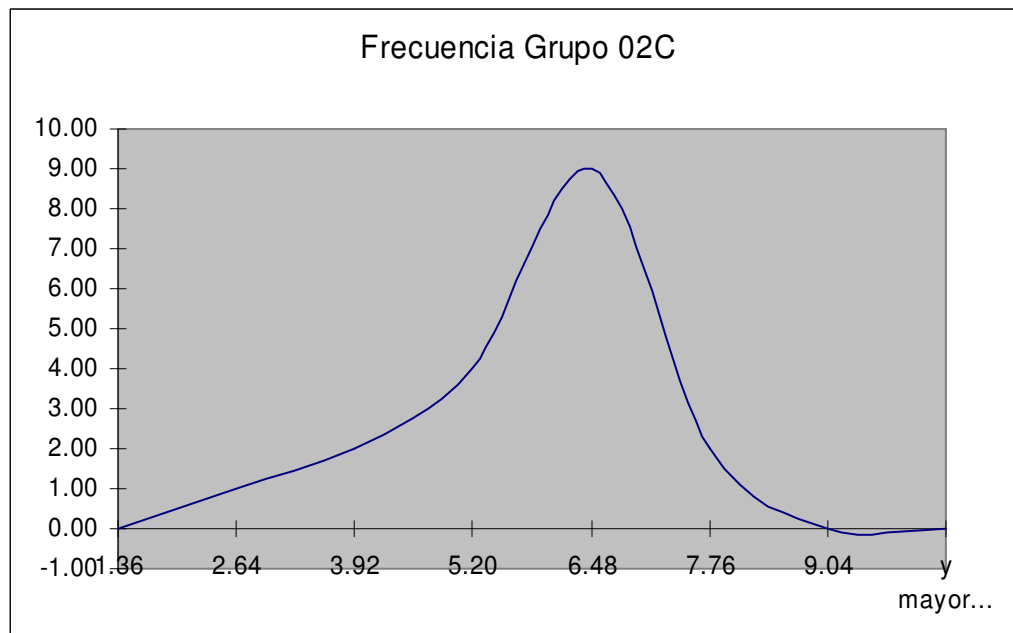


Figura 84. Diagrama de frecuencias del grupo experimental

En la versión final del modelo teórico están ya incluidas mejoras para elevar los resultados de estos tres factores.

En la figura 84 se muestra el diagrama de frecuencias del grupo experimental en el que se observa que hay un ligero sesgo hacia la derecha lo cual representa que hay una tendencia a obtener resultados más altos.

5.2.2.2. Resultados del Grupo de Control (04c)

El grupo de control obtuvo un índice global de 74 puntos en la prueba de HaDiCA, el cual se detalla en la tabla 54.

Grupo: 04C

Total: 74.00

ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
	5.79	6.00	6.21	3.47	3.05	4.53
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
	5.68	3.37	3.47	4.42	3.89	5.26
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
	3.79	2.84	2.95	2.95	2.95	3.37
TOTALES	ANÁLISIS	29.05	DISEÑO	26.11	CODIFICA	18.84

Tabla 54. Índice HaDiCA grupo de control

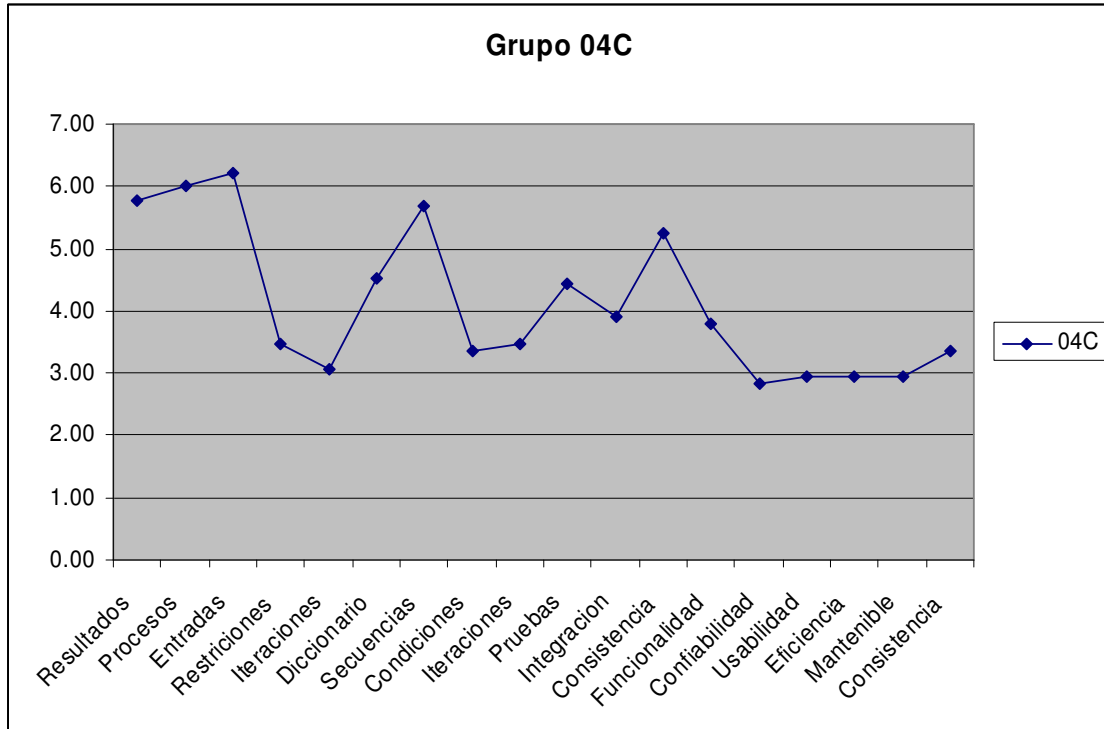


Figura 85. Histograma de habilidades específicas grupo de control

Cómo puede observarse el resultado más alto se obtuvo en la fase de análisis con 29.05 puntos, seguido de la fase de diseño con 26.11. La fase con menor puntaje es la de codificación con tan sólo 18.84 puntos. Esta fase en general obtuvo bajos resultados en todos sus componentes lo que demuestra consistencia pero a la baja. Por otra parte, el área mejor evaluada es la identificación de entradas seguida de la identificación de procesos. Esto se puede interpretar como resultado de que la forma de pensar en el análisis no fue en estrategia hacia atrás (backwards) buscando primero encontrar los resultados para de ahí derivar todo lo demás. Al parecer el modelo mental que se construyó fue el de hacia delante (forward) pensando primero en las entradas y luego tratar de descifrar que hacer con ellas.

En la figura 85 se muestra el histograma de los valores obtenidos en cada fase en el grupo de control (04c). Como se puede observar existe una correlación entre una baja habilidad para identificar restricciones e iteraciones, situación que concluye en bajos resultados en el diseño de las mismas y bajos puntajes en la construcción misma del código.

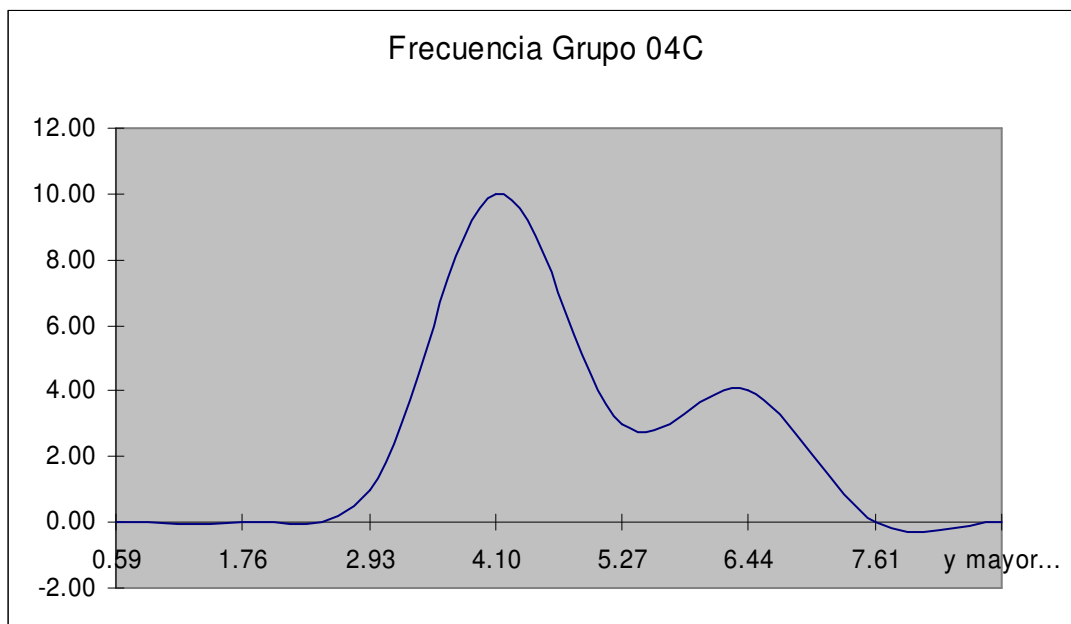


Figura 86. Diagrama de frecuencias de grupo de control

En la figura 86 se puede observar el diagrama de frecuencias del grupo de control. Se puede observar que el grueso del grupo tiene evaluaciones medias mientras que un pequeño grupo tiene resultados mejores que los demás.

5.2.2.3. Análisis Comparativo

En la tabla 55 se muestran los valores que se obtuvieron por cada factor tanto en el grupo de experimental (02c) como el de control (04c). La tercera columna muestra la diferencia en la que se observa que en todos los rubros el grupo experimental tiene una diferencia positiva con excepción del diseño de pruebas.

Indicador	Grupo 02C	Grupo 04C	Diferencia
Resultados	7.50	5.79	1.71
Procesos	6.65	6.00	0.65
Entradas	6.40	6.21	0.19
Restricciones	5.60	3.47	2.13
Repeticiones	3.10	3.05	0.05
Diccionario Datos	5.90	4.53	1.37
Total análisis	35.15	29.05	6.1
Secuencias	6.30	5.68	0.62
Condiciones	5.10	3.37	1.73
Repeticiones	3.65	3.47	0.18
Pruebas	2.10	4.42	-2.32
Integración	5.20	3.89	1.31
Consistencia	5.45	5.26	0.19
Total de diseño	27.80	26.11	1.69
Funcionalidad	5.70	3.79	1.91
Confiabilidad	5.30	2.84	2.46
Usabilidad	5.25	2.95	2.3
Eficiencia	4.45	2.95	1.5
Mantenible	5.25	2.95	2.3
Consistencia	5.15	3.37	1.78
Total de codig.	31.10	18.84	12.26
Índice total	94.05	74.00	20.05

Tabla 55. Comparativo de índice HaDiCA

En la tabla 56 se presentan los datos de análisis estadístico descriptivo en forma comparativa entre en grupo experimental (02c) y el de control (04c). Es interesante observar que la media, mediana y moda del grupo experimental (02c) están muy armonizadas mientras que en el grupo de control existe una diferencia de más de una desviación estándar entre la media y la moda.

Concepto	Grupo 02c	Grupo 04c
Media	5.225	4.11111111
Error típico	0.30252883	0.27843151
Mediana	5.275	3.63157895
Moda	5.25	2.94736842
Desviación estándar	1.28352112	1.18128485
Varianza de la muestra	1.64742647	1.39543389
Curtosis	1.22742916	-1.10205538
Coefficiente de asimetría	-0.8286077	0.67867574
Rango	5.4	3.36842105
Mínimo	2.1	2.84210526
Máximo	7.5	6.21052632
Suma	94.05	74
Cuenta	18	18
Mayor (1)	7.5	6.21052632
Menor(1)	2.1	2.84210526
Nivel de confianza (95.0%)	0.63828003	0.58743913

Tabla 56. Análisis estadístico comparativo

En la figura 87 se observan los diagramas de frecuencia comparativo entre el grupo 02c con el 04c. Preliminarmente se puede observar que la diferencia de las medias muestrales entre ambas distribuciones se calcula:

Media de 02c (Grupo experimental) = 5.225

Media de 04c (Grupo de control) = 4.111

Diferencia entre medias = 1.114

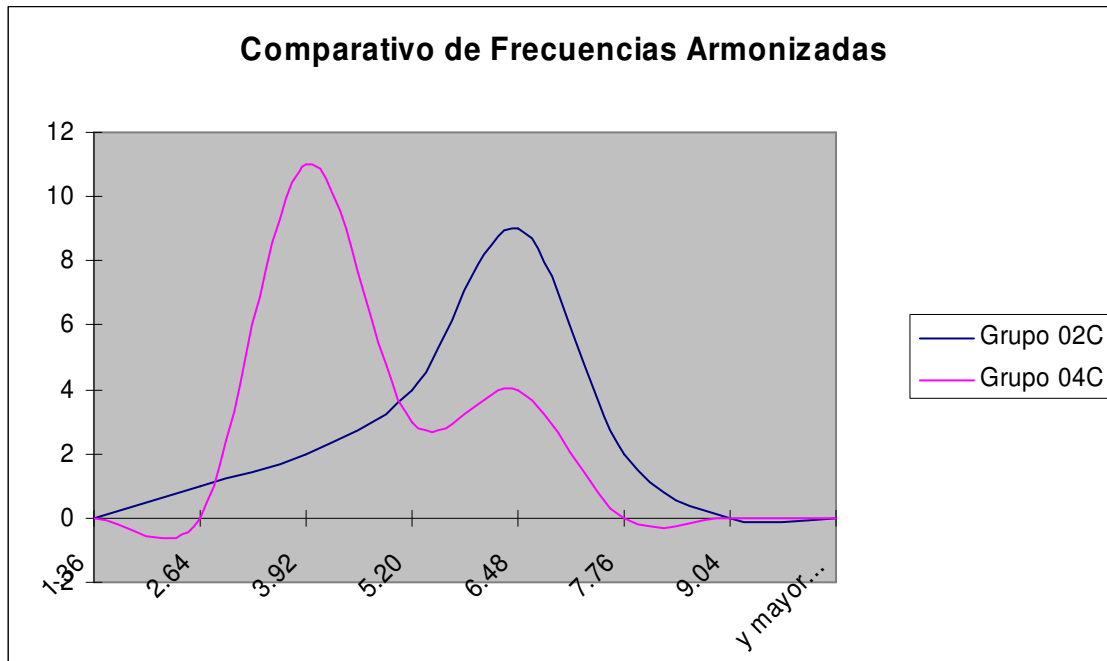


Figura 87. Diagrama de frecuencias comparativo

En la figura 88 se observa la diferencia acumulada de los índices de HaDiCA entre los grupos experimental y de control en el que se nota un incremento de 21.32% del grupo experimental con respecto al grupo de control en su rendimiento.

En la figura 89 se muestra el perfil comparativo entre los grupos experimental y de control. El área bajo el polígono representa el impacto. Como puede observarse, el grupo experimental tiene una mayor cobertura que el de control concluyendo que tiene un mayor impacto en la competencia de la programación.

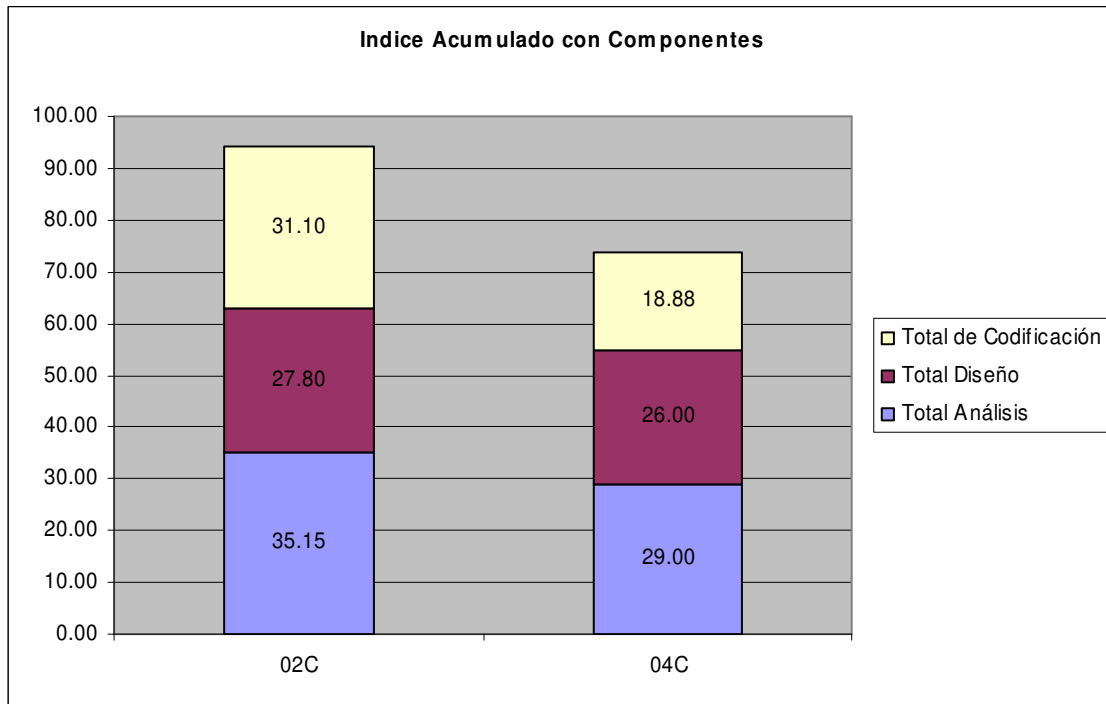


Figura 88. Comparativo de índices

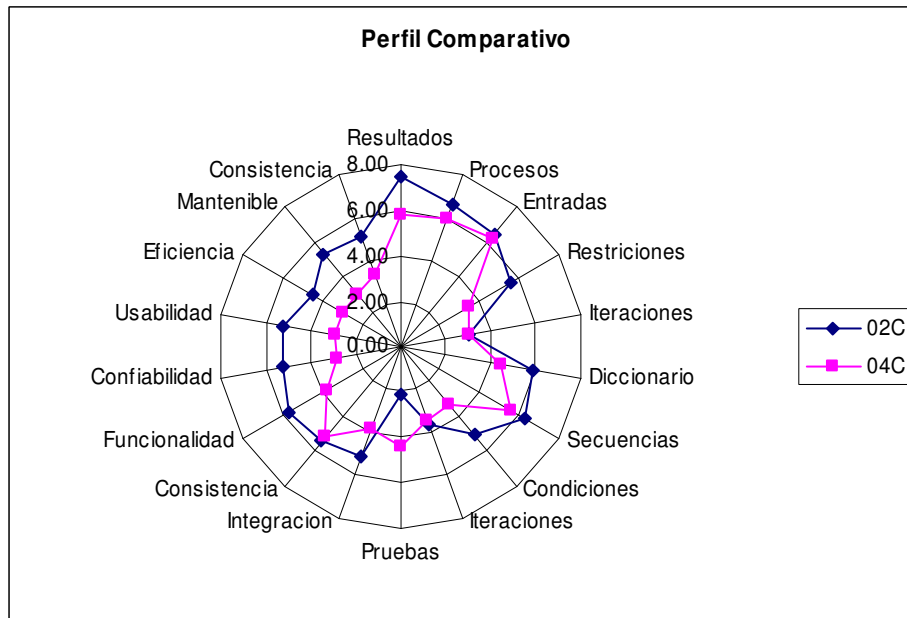


Figura 89. Perfil comparativo

5.2.3. Prueba de Hipótesis

De acuerdo al diseño del experimento definido en el capítulo uno, la aceptación o rechazo de la hipótesis nula depende de si hay diferencia estadística significativa entre Y_a (G_e) y Y_a (G_c). Se ha decidido aplicar la prueba t de *student* debido a que se trata de un análisis de muestras

pequeñas (menos de 30 observaciones). Esta prueba requiere definir el nivel de confianza (alpha) de la prueba, típicamente se aplica un alpha de 0.05, sin embargo, se ha decidido incrementar el nivel de confianza para disminuir la probabilidad del error tipo I (rechazar la hipótesis nula cuando esta debió ser aceptada) al mínimo, por lo tanto el nivel de confianza se ha llevado a 0.01.

Debido a que la prueba HaDiCA consta de 18 indicadores, los grados de libertad de la comparación de las dos muestras es de 34 lo cual nos arroja un valor crítico de t de 2.7283. La hipótesis nula será rechazada, y aceptada la alterna, solamente si el valor de la prueba t es superior al valor crítico. En la tabla 57 se observan los resultados obtenidos en el cálculo.

Prueba t para dos muestras suponiendo varianzas desiguales		
Alpha = 0.01	Grupo 02c	Grupo 04c
	$Y_a (G_e)$	$Y_a (G_c)$
Media	5.225	4.10416667
Varianza	1.64742647	1.383221
Observaciones	18	18
Diferencia hipotética de las medias	0	
Grados de libertad	34	
Estadístico t	2.73155268	
$P(T \leq t)$ una cola	0.00496102	
Valor crítico de t (una cola)	2.44114961	
$P(T \leq t)$ dos colas	0.00992203	
Valor crítico de t (dos colas)	2.72839436	

Tabla 57. Prueba t de *student* de índices HaDiCA entre grupos G_e y G_c

Como se puede observar el valor de t es de 2.73155268 y es superior al valor crítico de 2.728394 así que, incluso con una alpha de 0.01, si hay diferencia estadística significativa entre Y_a del Grupo Experimental y Y_a del grupo de control por lo que se concluye que:

Se descarta la hipótesis nula y se acepta la alterna por lo tanto, el modelo de Construcción de Algoritmos Apoyado en Heurísticas **Si** mejora las habilidades humanas para diseñar y construir algoritmos imperativos.

5.3 Alcance de objetivos

Al inicio de este trabajo se definieron los objetivos en la sección 1.4. Después de realizar los trabajos programados se puede establecer el alcance que se tuvo de ellos. A continuación se retoman tanto los objetivos principales como los específicos para hacer un análisis.

5.3.1. De los Objetivos Generales.

Los objetivos generales definidos son dos:

1. Crear un modelo de Construcción de Algoritmos Apoyado en Heurísticas que contemple de forma integral el trípode de la programación para que mejore la habilidad humana en el diseño y construcción de algoritmos imperativos.

El modelo CAAH se describe en su totalidad en el capítulo 3 en el cual se detalla cada uno de sus componentes en sus dimensiones teóricas, humanas, herramientas y heurísticas. Este modelo es original en su concepción y diseño a pesar de reutilizar todos los conocimientos, herramientas, técnicas y demás postulados que a lo largo de los años los estudiosos de las ciencias de la computación han aportado. Se puede decir que el objetivo trazado fue alcanzado en su totalidad.

2. Diseñar y aplicar un experimento de campo que pruebe que efectivamente el modelo de Construcción de Algoritmos Apoyado en Heurísticas mejora la habilidad humana para el diseño y construcción de algoritmos imperativos.

Crear un modelo teórico sin un escenario en el cual se demuestre su validez es tener un modelo incompleto. Por ello el segundo gran objetivo de este trabajo fue probar las potencialidades que el modelo aporta, estudiarlo en campo, retroalimentarse de los resultados y perfeccionarlo. En el capítulo 4 se describe con detalle el proceso de diseño y aplicación del experimento probando la primera versión del modelo CAAH. Durante los resultados de su aplicación se descubrió que las fases de pruebas estaban débiles y en consecuencia se reforzaron tal y como se puede ver en el diseño actual. El nivel de rigor y aplicación metódica del experimento le aporta un alto grado de confiabilidad. Durante su ejecución se aplicaron 28 criterios de validación divididos en cinco categorías. Con estos elementos podemos concluir que el segundo gran objetivo de esta Tesis también fue alcanzado.

El diseño del modelo teórico y el diseño del experimento son objetivos pragmáticos alcanzados, sin embargo, un objetivo global de alto interés es que en verdad sea funcional y útil este modelo. El resultado de la prueba de hipótesis explicado en la sección 5.1.3 demuestra, a través de análisis estadístico con α de 0.01, que el modelo CAAH si mejora las habilidades humanas para construir algoritmos lo cuales un objetivo alcanzado de gran importancia.

5.3.2. De los Objetivos Específicos.

Los objetivos generales fueron divididos en cinco específicos que a continuación se retoman y analizan.

1. Diseñar una metodología de desarrollo de habilidades de programación.

A pesar de que mucho se cita en la bibliografía la importancia de las habilidades de la programación, la ausencia de detalles sobre cuales son estas habilidades, como se clasifican y como se desarrollan es notoria. Por ello, y como parte del modelo holístico CAAH, uno de sus pilares es justamente, dentro del marco humano, el estudio de las habilidades y su desarrollo. En la sección 3.6 se aborda con detalle un modelo tridimensional de la inteligencia basado en los trabajos de Guilford y utilizando esta plataforma se identifican las habilidades intelectuales fundamentales para cada fase de una metodología de microingeniería de software. Una vez

identificadas las habilidades se clasifican en una taxonomía y se propone un modelo para su desarrollo. Considerando todo esto se puede concluir que este objetivo ha sido alcanzado.

2. Diseñar una metodología de microingeniería de software.

La ingeniería de software ha avanzado mucho desde que dio sus primeros pasos en los 70's. Los modelos actuales cuentan con toda una colección de principios, lenguajes y herramientas para poder transitar del dominio del problema al dominio de la solución. Sin embargo, todos estos modelos se concentran en los niveles de abstracción más altos del problema dejando sin detallar aspectos específicos del modelado algorítmico que requerirán, sin duda, al interior de las clases o componentes de software diseñados. La metodología de microingeniería de software se concentra, justamente, en esta área complementando las actuales metodologías. La sección 3.4 de este trabajo aborda con detalle cada fase, etapa, producto, herramienta y habilidad que es necesaria en el proceso de modelado algorítmico. Partiendo del nivel de detalle que se ha descrito sobre este particular se determina que este objetivo ha sido logrado.

3. Diseñar un lenguaje visual de modelado imperativo.

La necesidad de contar con herramientas visuales de modelado imperativo ha existido desde el nacimiento de la programación como puede notarse en la investigación del estado del arte del capítulo 2. Diversos problemas aquejan a los actuales modelos como se explica en la sección 3.5.1.3 por lo que este trabajo aporta un lenguaje visual de modelado imperativo denominado LeMVI que se describe con detalle en la sección 3.5.2. Este lenguaje aspira a resolver varios de los problemas actuales actuando en combinación con el resto de los elementos del modelo CAAH. Considerando que el diseño del lenguaje está definido en la sección ya citada se valora que este objetivo ha sido cubierto.

4. Crear un conjunto de heurísticas de diseño algorítmico.

Las heurísticas han sido definidas como un conjunto de reglas que orientan a un ser humano sobre el proceso de análisis, diseño y construcción de algoritmos. Este proceso es de tipo progresivo, iterativo e incremental. La aportación de un conjunto compacto y práctico de heurísticas se describe en la sección 3.3 de esta tesis. Cada una de ellas se concentra en aspectos particulares de la metodología de microingeniería de software buscando facilitar el proceso de descubrimiento del algoritmo. Partiendo del principio de que, además de haberse descrito teóricamente en la sección ya citada, se ejemplifica su uso y aplicación a través de la descripción de un caso de estudio explicado en la sección 3.4.7 se concluye que este objetivo ha sido alcanzado.

5. Diseñar una arquitectura para la creación de sistemas de Diseño de Algoritmos Asistido por Computadora que pueda extender las capacidades de los CASE actuales.

Si bien es cierto que el trabajo intelectual es fundamental en el proceso de análisis, diseño y construcción de algoritmos las herramientas de asistencia por computadora han sido una realidad desde hace ya varios años. Estas herramientas de tipo CASE operan en el marco de una metodología y un lenguaje de modelado, el más aceptado en la actualidad es el UML. Esta tesis ha propuesto una metodología y un lenguaje para ampliar las fronteras de modelado actual. La

definición de una arquitectura que aporte claridad sobre como desarrollar una herramienta de software que facilite el proceso de microingeniería de software y que pueda conectarse con los actuales sistemas CASE es uno de los objetivos planteados, mismo que ha sido cubierto con las definiciones detalladas en la sección 3.5.3 de este trabajo.

En conclusión, se puede decir que todos y cada uno de los objetivos tanto generales como específicos de esta tesis han sido alcanzados y desarrollados al interior de los capítulos dos, tres y cuatro. El capítulo dos concentrándose en el estado del arte en cada uno de los tres dominios de interés: teórico, humano y herramental; el tres describiendo con detalle el modelo completo incluyendo un caso de estudio; y el cuatro narrando la crónica del diseño y aplicación del experimento de campo.

5.4 Aportaciones

Diversas son las aportaciones que arroja este trabajo. El Modelo CAAH utiliza muchos componentes previamente elaborados a lo largo del tiempo por distintos estudiosos de la tecnologías de información, pero también utiliza muchos componentes nuevos creados específicamente para esta investigación. En la tabla 49 de la sección 5.1 se han señalado varias de ellas además de las explicaciones detalladas al interior del cuerpo de la tesis.

La concepción misma del modelo, su análisis, su ensamblaje con piezas nuevas y previas son contribuciones generales que se apoyan sobre aportaciones más específicas como las metodologías internas del modelo hasta aspectos puntuales como métodos, herramientas, productos, heurísticas y habilidades específicas para cada fase de esta visión holística expuesta.

Con la intención de puntualizar las aportaciones de mayor trascendencia en este trabajo se enumeran en forma resumida en las siguientes secciones.

5.4.1. Dentro del Marco Teórico:

El marco teórico abordó los métodos, principios, técnicas, herramientas y demás requerimientos necesarios para el análisis, diseño y construcción de algoritmos. Las aportaciones de mayor importancia en esta sección son las siguientes:

- Metodología de Microingeniería de Software.- Metodología iterativa e incremental que incluye fases, productos y herramientas claras para cada etapa del proceso de descubrimiento del algoritmo así como los procedimientos de transición entre fase y fase. En su interior se encuentran métodos y herramientas así como momentos de aplicación de las heurísticas y de las habilidades mentales identificadas.
- Heurísticas de Diseño de Algoritmos.- Conjunto de doce reglas, llamadas de lógica de programación, que orientan en el proceso de diseño de un algoritmo para la resolución de problemas computables, donde los diseños se descubren por la evaluación del progreso logrado en la búsqueda de un algoritmo final. También incluye la definición de términos utilizados por las reglas

Otras aportaciones más específicas son:

- Como parte de los métodos y herramientas a aplicar dentro de la microingeniería se encuentran: Método de análisis de microingeniería hacia atrás, método de tránsito de análisis a diseño, método de esqueletos, método de pruebas secuencia-condición-repetición y árbol de descomposición de procesos.
- Como parte de los productos originales resultado de aplicar microingeniería se encuentran: Tablas de resultados, listados de procesos, tablas de entradas, relaciones de restricciones de l mundo real y del mundo virtual y relación de repeticiones.

5.4.2. Dentro del Marco del Talento Humano:

El marco del talento humano abordó un análisis de la inteligencia basado en los estudios de Guilford. Partiendo de este punto se realizaron estudios para identificar, clasificar y desarrollar las habilidades intelectuales fundamentales involucradas con la competencia de la programación. Las aportaciones de mayor importancia son:

- Taxonomía de habilidades.- Conjunto de habilidades mentales específicamente relacionadas con la competencia de la programación clasificadas y definidas para facilitar su identificación y desarrollo.
- Modelo de desarrollo de habilidades.- Metodología propuesta para desarrollar, fortalecer y evaluar las habilidades mentales relacionadas con la competencia de las programación e identificadas en la taxonomía citada. La estrategia se presenta también en forma de espiral incremental.

Otras aportaciones más específicas son:

- Identificación del universo de las habilidades intelectuales utilizadas para programar.
- Análisis estadístico del uso de las habilidades por fase dentro del ciclo de vida del software.
- Identificación de habilidades de alto impacto a través de la aplicación de la ley de Parto.
- Ubicación de las habilidades dentro de un proceso de microingeniería de software como estrategia de desarrollo.

5.4.3. Dentro del Marco Herramental

El marco herramental aborda elementos de apoyo al modelo CAAH. Durante la presenta investigación se analizaron las distintas herramientas y lenguajes que se han utilizado a lo largo de los años para el modelado algorítmico y la programación. Se estudiaron los beneficios, problemáticas y limitaciones de cada uno. Con esta experiencia se propusieron, como principales aportaciones en esta área las siguientes herramientas:

- Lenguaje de modelado visual imperativo (LeMVI).- Lenguaje visual propuesto para realizar el modelado lógico de un algoritmo. Compuesto de dos capas, una abstracta y una concreta.
- Arquitectura DAAC.- Arquitectura que define las responsabilidades e interacciones entre los módulos necesarios propuestos para poder construir una herramienta para hacer Diseño de Algoritmos Asistido por computadora.

Otras aportaciones más específicas son:

- Análisis de las problemáticas de visibilidad, mantenibilidad, complejidad, ambigüedad, libertad estructural, legibilidad y productividad.
- Conjunto de estrategias de solución a las problemáticas ya citadas.
- Aplicación de pruebas de validación visual de LeMVI en campo.

5.4.4. Dentro del Contexto Experimental

Un modelo teórico tiene validez en la medida que puede ser demostrado y aplicado. Por ello, durante esta investigación se aplicó un experimento de campo que aportara información sobre la pertinencia, aplicabilidad y utilidad de los principios que forman el modelo. Las aportaciones de mayor importancia en esta fase son:

- Experimento de Campo.- En el que se demuestra que los resultados entre el grupo de control y el experimental tiene diferencia estadística significativa con un alpha de 0.01 comprobando que el modelo en realidad aporta mejora observable al ser aplicado.
- Prueba de habilidades (HaDiCA).- Prueba para evaluar la habilidad de diseño y construcción de algoritmos. La prueba incluye ejercicios, reglas de aplicación, rubricas de evaluación y tabulación y formatos gráficos y numéricos de interpretación que permiten combinar el área cualitativa y la cuantitativa.

Otras aportaciones más específicas son:

- Conjunto de escenarios de PBL tanto para aplicación de pruebas como para casos de estudio.
- Análisis estadístico de las distribuciones identificando áreas de fortaleza y de oportunidad al interior del modelo CAAH.
- Índice HaDICA que mide el grado de desarrollo que tiene la habilidad superior de la programación en las personas.
- Diagramas de perfil basado en 18 descriptores inmersos en procesos de análisis, diseño y construcción.
- Integración de 28 criterios divididos en cinco categorías de validez experimental
- Análisis estadísticos de diversas pruebas *t* de *student* para probar desde homogeneidad grupal al inicio del experimento hasta diferencias estadísticas significativas para pruebas de hipótesis.

5.4.5. Otros Beneficios y Aportaciones

La simple descripción de un modelo no siempre es suficiente para entender las sutiles relaciones que cada componente puede tener con los demás, incluso la prueba experimental tampoco describe las interrelaciones, vínculos y demás conexiones entre componentes, métodos, herramientas, heurísticas y habilidades. Por ello, y con la intención de clarificar estos procesos, otra importante aportación de este trabajo es la aplicación del siguiente:

- Caso de Estudio.- En el que se aplica completa la metodología en tres iteraciones para mostrar prácticamente la relación entre la metodología de microingeniería, las heurísticas de diseño y la aplicación de habilidades a través del uso de un lenguaje de modelado visual. Todo esto mientras se resuelve un escenario simple pero que aporta posibilidades de estudio en cada fase metodológica.

Además de todos los componentes propios del modelo (teorías, procedimientos, herramientas, etc.) se identifican tres tipos de aportaciones adicionales que en forma genérica se describen a continuación.

Social

- Mejores procedimientos educativos de la programación.
- Mejoras en las habilidades de programación de los estudiantes

Profesional.

- Mejores prácticas de ingeniería de software.
- Disminución de índices de reprobación y abandono de la carrera.

Científica

- Mejor comprensión de las habilidades humanas de la programación.
- Mejores métodos para medir las habilidades de programación.
- Mejores herramientas para la construcción de software.

Otros beneficios y aplicaciones que se han identificado son los siguientes:

1. Formalizar los mecanismos para desarrollar las habilidades humanas para la construcción de algoritmos tiene un impacto directo en la calidad del Software. Esto redundará en mejorar las expectativas de éxito en los proyectos de software.
2. Conocer las habilidades humanas involucradas en el proceso de la programación permite a los investigadores del área de inteligencia artificial crear espacios de estudio sobre el desarrollo artificial de estas habilidades.
3. Los procesos educativos y formativos de futuros ingenieros se perfeccionan y se alcanzan mejores índices de aprovechamiento, disminuye la deserción y crea confianza en las habilidades de programación de los estudiantes.
4. Al incorporar técnicas y principios de ingeniería de software en etapas tempranas, como es la de aprender a programar, brinda una continuidad y familiaridad de los estudiantes para con estas materias. Cuando lleguen a cursar los temas más avanzados de ingeniería ya todo es conocido y potencia los alcances de estas materias.

5. Al hacer énfasis en los procesos de análisis y diseño tanto en los modelos teóricos como de habilidades crea hábitos de trabajo en los futuros ingenieros de hacer esfuerzos conscientes e intencionados en estas fases evitando el clásico enfoque de pretender obviar etapas e ir directo a la programación, acción fomentada justamente por los actuales procesos educativos de la programación y que desemboca en malas prácticas de ingeniería.
6. La creación de código automática siempre ha sido una aspiración. El contar con herramientas visuales a diferentes niveles de abstracción permite crear el camino para alcanzar esta meta.
7. Durante años, la portabilidad ha inspirado lemas tales como “programa una vez, ejecútalo en muchas plataformas”. El contar con herramientas de diseño imperativo y una arquitectura de tipo CASE que permita a la postre la creación de código automático podrá crear nuevos lemas al estilo: “Diseña una vez, impleméntalo en muchos lenguajes y muchas plataformas”.

Por último, una aportación más de este trabajo de tesis es la publicación de 14 trabajos distribuidos de la siguiente forma:

- Dos publicaciones en revistas científicas arbitradas y con registro de CONACYT.
- Un informe técnico del estado del arte en materia de diseño y modelado algorítmico.
- Once ponencias en congresos nacionales e internacionales.

El detalle de estas publicaciones puede encontrarse en la sección correspondiente al final de esta tesis.

5.5 Trabajos Futuros

Este trabajo es producto de una problemática internacional que genera una gran preocupación en las esferas académicas del área de informática y computación así como de la calidad del software en áreas de ingeniería.

El alcance de esta tesis se limitó a la definición del modelo con sus tres dimensiones, además de presentar un caso de estudio y un experimento de campo. El experimento de campo arrojó luz valiosa para reflexionar sobre las debilidades del modelo y crear una segunda versión en la que se apuntalan las debilidades identificadas. Sin embargo, no se puede considerar que el trabajo haya concluido. Como todo trabajo de investigación, el mismo no está agotado y hay nuevos retos que resolver.

Con un verdadero espíritu de investigación, a esta primera aplicación del experimento deben seguir otras más en las que se pueda aprender de cada una de ellas y refinar el modelo perfeccionándolo y alcanzando el verdadero objetivo final: Mejorar los procesos de programación. Por ello, ya están en marcha las siguientes fases de un plan más ambicioso que está formado por cinco fases dónde la primera se ha explicado en este trabajo y las siguientes se describen a continuación:

2ª Etapa, nueva prueba en campus incluyendo las modificaciones (septiembre 2006).- Se repetirá la aplicación del experimento de nuevo al interior del Campus Hispano utilizando el mismo diseño de experimento y las mismas reglas. Si, como se espera, los datos son mejores que en el primer experimento se pasará a la 3ª etapa de inmediato. Si no resultasen aún mejores, se realizará un análisis de los resultados obtenidos y se aplicará la segunda etapa un ciclo lectivo después para poder incorporar los aprendizajes.

3ª Etapa, prueba inter-campus (enero 2007).- Una vez superada la segunda etapa ya está pactada con el Campus Querétaro y San Rafael de la UVM la aplicación del experimento para conocer los resultados en escenarios distintos. Se aplicará la misma política que en la etapa anterior para determinar si se prosigue a la 4ª etapa de inmediato o un ciclo después.

4ª Etapa, prueba inter-universitaria internacional (septiembre 2007).- En esta etapa se aplicará el experimento en el extranjero. Ya se iniciaron las negociaciones con las universidades Rey Juan Carlos y Europea de Madrid de España y con la de Buenos Aires en Argentina.

5ª Etapa, liberación.- Se entregará el modelo a las universidades que están colaborando en este proyecto para que hagan uso del modelo libremente en sus clases.

Todas estas Etapas ya están programadas debido a que esta investigación ha sido incluida en el Programa Institucional de Investigación en Tecnociencias (PIIT) de la Universidad del Valle de México dictaminado durante febrero de 2006 y asumirá los costos de las aplicaciones de las siguientes etapas.

Otros trabajos futuros que emanan de este trabajo son los siguientes:

- Construir un software bajo la arquitectura DAAC (como tesis de maestría de alumnos del CIC-IPN). Una vez que se cuenta con la arquitectura propuesta, el siguiente paso es construir el software.
- Ampliación de LeMVI para convertirlo en un LVP completo (como tesis de maestría de alumnos del CIC-IPN). LeMVI puede ser utilizado a simple papel y lápiz, sin embargo, la aspiración siguiente es que se convierta en un LVP completo automatizado. Dentro de la Arquitectura DAAC se consideran módulos específicos para su implementación.
- Integración de un CASE con el sistema construido bajo la arquitectura DAAC (Como tesis de maestría de alumnos del CIC-IPN). Las fronteras de los CASE actuales pueden ampliarse hasta el modelado algorítmico de forma tal que pueda crearse ingeniería de software hace adelante completa (Full Forward Engineering). La elaboración de la interfaz entre los actuales CASE y el software bajo la arquitectura DAAC se antoja un proyecto futuro interesante.
- Construcción de una biblioteca de patrones de diseño algorítmico imperativo. El estudio y diseño de patrones se ha convertido en una disciplina que porta ventajas en muchos sentidos. La posibilidad de crear una línea de investigación sobre patrones algorítmicos es un esfuerzo que enriquecería las potencialidades del modelo.
- Cruce Conocimiento-Habilidad de las materias de introducción a la programación. Este trabajo realizó un cruzamiento entre las etapas metodológicas de microingeniería de software y las habilidades intelectuales fundamentales. Sin embargo, los modelos de competencias también incluyen el área cognitiva por lo que realizar un estudio para

identificar las habilidades por conocimiento señalado en planes y programas de estudio y/o libros de texto se prevé que puede ser de gran aporte futuro.

- Plan de solución de errores de programación. A partir de la colaboración con la Universidad de Buenos Aires, Argentina, elaborar un trabajo para vincular su investigación sobre los errores de programación con soluciones a través de la aplicación del modelo CAAH se estima de mutuo enriquecimiento.

5.6 Palabras Finales

El factor humano siempre ha sido un componente crucial en el éxito de todo proyecto, por supuesto en los proyectos de software también lo es. La búsqueda de mejores prácticas de ingeniería necesariamente tiene que considerar estrategias de talento como parte integral de su enfoque.

Al elaborar algoritmos a través de una metodología con pasos claros, en espiral e incremental se está formalizando una estructura mental en los futuros ingenieros lo cual permitirá abordar problemas más complejos y ascender en niveles de abstracción a través de un tránsito natural a la ingeniería de software perfeccionando la competencia de la programación y preparándolos para modelos de calidad del software.

Después del tiempo invertido y gran esfuerzo realizado para buscar toda clase de acciones realizadas históricamente para la mejora de la ingeniería de software y la formación de ingenieros; el haber diseñado, en forma integral, el Modelo de Construcción de Algoritmos Apoyado en Heurísticas; el haber realizado un caso de estudio y un experimento de campo con gran esperanza pero con la incertidumbre que todo experimento conlleva; al volver la vista atrás, esta tesis no deja más que la satisfacción de haber hecho un buen trabajo y haber aportado, aunque sea solo un poco, en el conocimiento sobre el fenómeno humano de la programación.

Publicaciones

Revistas:

- Vargas, R.; Gutiérrez, A. *Visión retrospectiva de los principios de la programación*. EPISTEME, Universidad del Valle de México, ISSN:1665-9317, diciembre 2004 (**arbitrada con registro de CONACYT**).
- Vargas, R.; Gutiérrez, A. *El Teorema de Jacopini en el Mundo de la programación orientada a Objetos*. EPISTEME, Universidad del Valle de México, ISSN:1665-9317, diciembre 2004, enero 2006 (**arbitrada con registro de CONACYT**).

Informe Técnico:

- Vargas, R.; Gutiérrez, A. *Visión retrospectiva del diseño de algoritmos*. Instituto Politécnico Nacional, Centro de Investigación en Computación. Reporte técnico, serie Azul No. 203. ISBN: 970-36-0219-3. 2004.

Congresos:

- Vargas, R.; Gutiérrez, A. *Lógica de Programación, Un enfoque Formal*. Congreso Nacional y Congreso Internacional de Informática y Computación ANIEI (CNCIIC 2002 Guadalajara).
- Vargas, R.; Gutiérrez, A. *Desarrollando la Lógica de Programación. Un enfoque formal*. 2ª Conferencia Iberoamericana en Sistemas Cibernética e Informática. Código de artículo: C338SO (CISCI 2003 Orlando).
- Vargas, R.; Gutiérrez, A.; Olivares, J. *Diseño de Algoritmos Asistido por Computadora*. Congreso Nacional y Congreso Internacional de Informática y Computación ANIEI. ISBN: Vol. 1 970-36-0101-4, Vol. 2, 970-36-0102-2 y 970-36-0100-6 (CNCIIC 2003 Zacatecas).
- Vargas, R.; Gutiérrez, A. *Visión retrospectiva de los principios de la programación y su impacto en la formación de Ingenieros y en la calidad del software*. Congreso Nacional y Congreso Internacional de Informática y Computación ANIEI. ISBN:970-36-0155-3 (CNCIIC 2004 Tepic).
- Vargas, R.; Gutiérrez, A. *Análisis Histórico del Diseño de Algoritmos*. 1er Congreso Institucional de Investigación, Innovación y Desarrollo de la Educación Media Superior y Superior. Universidad del Valle de México (CIIDE UVM 2005 México).
- Vargas, R.; Gutiérrez, A. *El Teorema de Jacopini en el Mundo de la programación orientada a Objetos*. 1er Congreso Institucional de Investigación, Innovación y Desarrollo de la Educación Media Superior y Superior. Universidad del valle de México (CIIDE UVM 2005 México).
- Vargas, R.; Gutiérrez, A. *Modelo de Algoritmos Basado en Heurísticas*. 1er Congreso Institucional de Investigación, Innovación y Desarrollo de la Educación Media Superior y Superior. Universidad del valle de México (CIIDE UVM 2005 México).

- Vargas, R.; Gutiérrez, A.; Suarez, S. *Heurísticas para la construcción de Algoritmos*. Congreso Nacional y Congreso Internacional de Informática y Computación ANIEI. ISBN: 970-31-0528-9 (CNCIIC 2005 Torreón).
- Vargas, R.; Gutiérrez, A. *Sistema de Evaluación de la Educación en Informática*. XII Jornadas de Enseñanza Universitaria de la Informática de AENUI. ISBN: 84-9732-545-1 (JENUI 2006 Bilbao).
- Vargas, R.; Gutiérrez, A.; Suarez, S. *Resultados de Aplicar un Modelo de Construcción de Algoritmos Apoyado en Heurísticas*. 1er Congreso Interdisciplinario de Investigación Aplicada de la Universidad del Valle de México UVM. (CIIA 2006 México).
- Vargas, R.; Gutiérrez, A.; Suarez, S. *Las Habilidades de la Programación*. 1er Congreso Interdisciplinario de Investigación Aplicada de la Universidad del Valle de México UVM. (CIIA 2006 México).

Referencias

- [1] *Actas de las XI Jornadas de la Enseñanza Universitaria de la Informática (JENUI 2005)*, Thomson, 2005.
- [2] Aho, A.; Sethi, R.; Ullman, J. *Compiladores principios, técnicas y herramientas*. Pearson Educación. México. pp. 169. 1990.
- [3] Alcalde, Eduardo; García, Miguel. *Metodología de la Programación. Aplicaciones en COBOL y Pascal*, McGraw Hill, pp. 205, 203, 207, 57, 25, 207, 6, 2, 219, 221, 387, 391, 360, 363, 214, 65, 21, 224, 12, 15. 1992.
- [4] Álvarez, L. *La Educación Basada en Competencias: Implicaciones, Retos y Perspectivas* Didac N. 36. Centro de Desarrollo Educativo. Universidad Iberoamericana.
- [5] Anderson, J.R., Corbett, A.T., Koedinger, K.R., & Pelletier, R., *Cognitive Tutors: Lessons Learned*, *Journal of the Learning Sciences*, 4(2), 167-207, 1995 en http://act.psy.cmu.edu/ACT/papers/Lessons_Learned.html
- [6] Ayala, G.; Yano, Y. *A Collaborative Learning Environment Based on Intelligent Agents*, paper submission to Expert Systems with Applications.
- [7] Barows, H.S.; Tamblyn, R.N. *Problem-based Learning*, Springer. 1980.
- [8] Beutelspacher, M., Franzoni, A. L. y Morales, A., *Tesis de ingeniería: Sistema de Apoyo Generalizado para la Enseñanza Individualizada (SAGE)*, ITAM, México, 1995.
- [9] Birtwistle, Graham M.; Dahal, Ole-Johan; Myhrhaug, Bjorn; Nygaard, Kristen. *Simula Begin*, Auerbach publishers. 1973.
- [10] Blackwell, A.; Whitley, K., Good, J.; Petre, M. *Programming in pictures, pictures of programs*. Discussion paper for the thinking with diagrams. 1997 interdisciplinary workshop (enero 1997), Portsmouth, Reino Unido.
- [11] Blanco, Isauro. *Hay más dentro de ti, El universo de la inteligencia*. Pearson Educación, pp. 179. 2002.
- [12] Böhm, Conrado; Jacopini, Giuseppe. *Flow diagrams, Turing Machines and languages with only two formation rules*. ACM, vol. 9 num. 5 mayo 1966.
- [13] Booch, Grady; Rumbaugh, James; Jacobson, Ivar. *El lenguaje Unificado de Modelado*. Addison Wesley. pp. 55. 1999.
- [14] Botoni, P. Formalizing Visual Languages. <http://kogs25.informatik.uni-hamburg.del~haarslev/v195www/talks>. 1995.
- [15] Brooks, Fredrick P. *No Silver Bullet*. Computer, pp. 10-19. Abril 1986.
- [16] Bruner, J., Acción, *Pensamiento y Lenguaje*, 2ª. Ed., Alianza Editorial Mexicana, 1986; 1ª ed. En español trad. Linaza, J. L., Madrid, 1984.
- [17] Budd, Timothy. *Introducción a la Programación Orientada a Objetos*. Addison Wesley, pp. 2, xvi, 20, xiii. 1992.
- [18] Burnett, M.; Baker; McIntyre. *Visual Programming*. Computer Vol. 28, No 3 pp 14-16. 1995.
- [19] Burns, H. L. y Capps, C.G., *Foundations of STI: An Introduction*, Polson, M. C, y Richardson, J. J. (editors), Foundations of Intelligent Tutoring Systems, Lawrence Erlbaum Associates, Estados Unidos, 1988.
- [20] Calloni, B. A. y Bagert, D. J. *ICONIC Programming in BACII Vs Textual Programming: Which is a Better Learning Environment?* <http://www.cs.ttu.edu/dept/research/bacii/sigcse94.html>. 1994.

- [21] Carey, Rikk; Bell, Gavin. *The Annotated VRML 2.0 Reference Manual*. Addison Wesley, 2004.
- [22] Catalina, Alonso. *Los estilos de aprendizaje: procedimientos de diagnóstico y mejora*, Ediciones Mensajero, Bilbao. pp. 104.
- [23] Cazau, Pablo. *Estilos de Aprendizaje: Generalidades*. CIIDET México.
- [24] Chag, S. Tortora; Yu, B. G. Guercio. A. *Icon Purity toward a formal theory of icons*. International journal of pattern recognition an artificial intelligence. World scientific publishing. Pp 377-392. 1987.
- [25] Colaboradores de Wikipedia. *Heurística*. Wikipedia, La enciclopedia libre, <http://es.wikipedia.org/w/index.php?title=Heur%C3%ADstica&oldid=3216152>
- [26] Cood, Edgar F. *A relational model of data for large shared data banks*. ACM 13 num. 6. junio 1970.
- [27] Copello, G.; Cataldi, Z.; Lage, F. *Interpretación de los errores que cometen los estudiantes en sus exámenes de algoritmia en su primer curso en la universidad*. Laboratorio de informática educativa y medios audiovisuales. Universidad de Buenos Aires. 2005 <http://www.fi.uba.ar/laboratorios/lie/lie.htm>
- [28] Cox, Brad J.; Novobilski, Andrew J. *Programación Orientada a Objetos. Un enfoque evolutivo*. Addison Wesley/Díaz de Santos. pp. xii. 1993.
- [29] Cox, Brad. *There is a silver bullet*. Byte. pp 209-218. octubre 1987.
- [30] Crotty, T., *Constructivist Theory unites Distance Learning and Teacher Education*, <http://edie.cprost.sfu.ca/~it/constructivist-learning.html>. Diciembre 1997.
- [31] Dale, E., *Logo Builds Thinking Skills*, en Run: Computer Education, Harper, D. O. y Stewart, J. H., Brooks/Cole Publishing Co., Estados Unidos, 1984.
- [32] Daves, Gordon B.; Gorgone, John T.; Couger, Daniel; Feinstein, David L; Longnecker, Herbert E. *IS'97 model curriculum and guidelines for undergraduate degree programs in information systems*. Association of Information Technology Professionals. 1007 <http://webfoot.csom.umn.edu/faculty/gdavis/curcomre.pdf>
- [33] Decker, Rick; Hirshfield, Stuart. *Pascal's Triangle. Reading, Written and Reasoning*, PWS-Keny, pp. xiv, 11. 1992.
- [34] DeMarco, Tom. *Structured Analysis and System Specification*. Yourdon Press, Nueva York. 1978.
- [35] Dijkstra, Edsger W.. *A Discipline of Programming*. Prentice Hall. pp. xiii. 1976.
- [36] Dromey, R. Geoff. *A model for software product quality*. IEEE Transactions on Software Engineering, vol. 21, No. 2 February 1995. pp 146-162.
- [37] Eduteka. *Matriz de Valoración: Rúbrica*. Fundación Gabriel Piedrahita Uribe.Cali - Colombia http://www.eduteka.org/ediciones/recomendado_julio02.htm
- [38] Fitter, M.; Green, T. *When do diagrams make a good computer language?* International Journal of man-machine studies, 11, pp. 255-261. 1978.
- [39] Forsythe, Alexandra I.; Keenan, Thomas A., Organick, Elliott I; Stenberg, Warren. *Lenguajes de Diagramas de Flujo*. Limusa. pp. 211. 1969.
- [40] Fowler, Martin. *UML distilled*, Addison Wesley Longman, pp 129. 1997.
- [41] Fraustro, Manuel. *Conocer: Consejo de Normalización y Certificación de Copetencia Laboral*. CONOCER, Ingenierías, Enero-Marzo 2000 Vol. III No. 7. <http://www.conocer.org.mx/>
http://ingenierias.uanl.mx/7/pdf/7_Manuel_Fraustro_Conocer.pdf#search=%22CONOCER%22

-
- [42] Gambo, R. y Reyes, A., *El uso de la computadora en la enseñanza de las matemáticas, documento para Proyecto de Investigación Educativa*, ITAM, México, 1997.
- [43] Gardner, Howard. *Inteligencias Múltiples*. Paidós Ibérica, Ediciones. 1997.
- [44] Geschke, C. M.; Mitchell, J. G. *On the problem of uniform references to data structures*, SIGPLAN notices, vol. 10, No. 6 junio 1975, pp. 31-42.
- [45] Glenn Brookshear, J. *Introducción a las ciencias de la computación*. Addison-Wesley Iberoamericana. pp. 141, 3, 151, 149, 148, 143, 4, 141. 1995.
- [46] Good, J. *VPLs and novice program comprehension: How do different languages compare?* Presentado para VL99 Japón. 1999.
- [47] Gosling, Peter E. *Programación Estructurada para microcomputadoras*, McGraw Hill, pp 3, xi, 1. 1985.
- [48] *Gran Diccionario Enciclopédico Ilustrado*. Selecciones del Reader's Digest. 1983.
- [49] Green, T.R.G. y Petre, M., *Usability Analysis of visual Programming Environments: A 'Cognitive Dimensions' Framework*. Journal of Visual Languages and Computing, 7(2), 1996.
- [50] Green, T.R.G., *Noddy's guide to Visual Programming*, Interfaces, British Computer Society Human-Computer Interaction Group, Reino Unido, 1995.
- [51] Guzmán Arenas, Adolfo. *Conferencia sobre las Calidad del Software y la Educación*. CIC-IPN. 2004.
- [52] Hoare, C.A.R. *Proof of correctness of data representations*, Acta informática, Vol. 1, 1972, pp. 271-281.
- [53] Jackson, M. A. *Principles of program design*, Academic Press, Londres, 1975.
- [54] Johnsonbaugh, Richard. *Discrete Mathematics*. Prentice Hall. pp. 143. 1997.
- [55] Joyanes Aguilar, Luis. *Metodología de la Programación. Diagramas de Flujo y programación estructurada*, McGraw Hill, pp 217, 9, 68, 219, 223, 225, 220, 39, 11. 1987.
- [56] Joyanes, L. *Metodología de la Programación. Diagramas de Flujo y programación estructurada*. McGraw Hill. pp. 38. 1987.
- [57] Kay, A., *User interface: a personal view*, en The art of human computer interface design. Laurel, B., Addison-Wesley. Estados Unidos, 1990.
- [58] Kendal, *Análisis y Diseño de sistemas*. Pearson. 1997.
- [59] Kernighan, Brian W.; Ritchie, Dennis M. *El Lenguaje de Programación C*, Pearson Educación, pp.4-5, 1991.
- [60] Kiper, J., Howard, E., Ames, Ch. *Criteria for evaluation of visual programming languages*. Journal of visual languages and computing. Vol. 8 No. 2 pp. 175-192. 1997.
- [61] Knuth, Donald. *The Art of Computer Programming, Vol. 1: fundamental algorithms*. Addison Wesley, 1997.
- [62] Larman, Craig. *UML y Patronos, Introducción al análisis y diseño orientado a objetos*. Prentice may, pp. 145. 1999.
- [63] Ledin, George Jr.; Ledin, Victor. *Manual de reglas para el programador*. Diana. pp 129. 1979.
- [64] Lemay, Laura. *Teach yourself Web publishing with HTML 4.0 in a Week*. Sams. 1997.
- [65] Levine, Guillermo. *Computación y programación moderna*. Addison Wesley. pp. 39, 42, 43, 176, 246. 2001.
- [66] Levine, Guillermo. *Introducción a la computación y a la programación estructurada*. McGraw-Hill. pp. 183, 214, 184, 201, 202. Año 1989.

- [67] Libro blanco, *Título de grado en Ingeniería Informática*, Agencia de Evaluación de la Calidad y Acreditación, Madrid, pp. 161, 162, 185.
- [68] Lizkov, Barbara H.; Zilles, Stephen N. *Programming with abstract data types*, SIGPLAN notices, 9, 4 abril 1974 pp. 50-59.
- [69] Marriot, K. Meyer, P. *Visual Language theory*. Springer verlag. 1998.
- [70] McMillan, Claude, González, Richard F. *Análisis de Sistemas*. Trillas. pp. 30. 1977.
- [71] Meyer, Bertrand. *Construcción de software orientado a objetos*. Prentice Hall. pp. 110. 1999.
- [72] Miller, P., et al, *Evolution of Novice Programming environments: The Structure Editors of Carnegie Mellon University*. Interactive Learning Environments, vol. 4, no. 2 1994.
- [73] Minsky, M., *Applying Artificial Intelligence to Education*, Computers and Communications: Implications for Education-Proceedings of Computer Technology in Education for 1985. Seidel, R. y Rubin, M., Academic Press, Estados Unidos, 1977.
- [74] Modelo Curriculares, comité de. *Modelos Curriculares Nivel Licenciatura Versión Actualizada*. Asociación Nacional de Instituciones de Educación en Informática. <http://www.aniei.org.mx>
- [75] Morris Mano, M.. *Lógica Digital y diseño de computadores*. Prentice Hall. pp. 28. 1982.
- [76] Oberlander, J.; Berna, P.; Cox, R.; Good, J. *The GRIP Project, or the match-mismatch conjecture and learning to use dataflow visual programming languages*. Universidad de Edimburgo y Universidad de Leeds. <http://www.cld.leeds.ac.uk/~paul/grip.html>. 1997.
- [77] Ole, Dahl; Dijkstra, Edsger; Hoare, Charles. *Structured Programming*, Academic Press, New York, 1972.
- [78] Parnas, David. *A technique for software module specification with examples*, Communications of the ACM, vol. 15, No 5. Mayo 1972, pp.330-336.
- [79] Pattis, Richard E. *Introducción Gradual a la programación. El Robot Karel*, Limusa. 1987.
- [80] Pékelis, V. *Pequeña Enciclopedia de la Gran Cibernética*. Editorial MIR, Moscú. pp. 273 1977.
- [81] Pérez; Jiménez. *Programación Neurolingüística y sus estilos de aprendizaje*. <http://www.aldeaeducativa.com/aldea/tareas2.asp?which=1683>. 2001.
- [82] Perry, Greg. *Aprendiendo principios de programación en 24 horas*. Prentice Hall. pp 68, 67, 84. 1999 .
- [83] Piaget, J. *Education et Instruction*, Francia, 1967, traducción al español de Acevedo, H., Ecuación e Instrucción, Editorial Proteo, Argentina, 1970.
- [84] Polya, George. *How to Solve It*, 2nd ed., Princeton University Press, 1957.
- [85] Poole, Philip. *PBL Insigh*. Center for Problem-Based Learning at Samford University. Vol. 6 <http://www.samford.edu/pbl/definitions.html>
- [86] Pressman, Roger S. *Ingeniería del Software. Un enfoque Práctico*. McGraw Hill. pp 224. 2001.
- [87] Pressman, Roger. *Ingeniería de Software, un enfoque práctico*, McGraw Hill, pp 249, 1998.
- [88] *Problem Based Learning*. Center for Teaching, learning ans Scholarship. Samford University. http://www.samford.edu/ctls/problem_based_learning.html
- [89] Revilla, Diana. *Estilos de aprendizaje*, Temas de Educación, Segundo Seminario Virtual del Dep. de Educación de la Pontificia Universidad Católica del Perú, <http://www.pucp.edu.pe/~temas/estilos.html>. 1998.

-
- [90] Scanlan, David. *Structured flowcharts outperform pseudocode: an experimental comparison*. IEEE software (6) USA, 1989.
- [91] Senn, James A. *Análisis y diseño de sistemas de información*. McGraw-Hill. pp 502, 505, 168. 1987.
- [92] Servicio Público de Empleo Estatal. *Resumen Panorámico de los sistemas de formación profesional en España*. Instituto de Empleo. Ministerios de Trabajo y Asuntos Sociales. España. <http://www.inem.es/otras/referNet/pdfs/sintesisfp.pdf>
- [93] Shelly, Gary B.; Cashman, Thomas J. *Introduction to Computers and data processing*. Anaheim Publishing. pp 502, 10.18, 11, 4. 1980.
- [94] Shneiderman, B. et. Al. *Experimental investigations of the utility of detailed flowcharts in programming*. Communications of the ACM, Volumen 20 número 6 junio de 1977.
- [95] Sleeman, D., *The Challenges of Teaching Computer Programming*, Communications of the ACM, September 1986, Volume 29, Number 9.
- [96] Soloway, E., *Learning to Program = Learning to Construct Mechanisms and Explanations*, Communications of the ACM, Volume 29, Number 9, Septiembre, 1986.
- [97] Sommerville, Ian. *Ingeniería de Software*. Pearson. 2001.
- [98] Sommerville, Ian. *Software Engineering*, Addison Wesley, pp 242, 2004.
- [99] Tinoco, M. *Educación Basada en Competencias en el Ámbito de la Educación Superior* ANUIES. Didac. N. 37. Centro de Desarrollo Educativo. Universidad Iberoamericana.
- [100] Travers, M., *Programming with Agents: New metaphors for thinking about computation*, tesis doctoral de MIT, en <http://lcs.www.media.mit.edu/people/mt/thesis/mt-thesis.html>
- [101] Vargas, Ricardo; Gutiérrez, Agustín. *La lógica de programación como habilidad para diseñar algoritmos: un enfoque formal*. CNIC 2003 y CIIC 2003, ANIEI 2003.
- [102] Voss, Greg. *Programación Orientada a Objetos: una introducción*. McGraw-Hill. pp. 9, 6, 23, 20. 1994.
- [103] Warnier, Jean D. *Entrainement a la programmation*. Constriction des programmes. 1975.
- [104] Wenger, E., *Artificial intelligence and tutoring systems*. Morgan Kaufman Publishers, Inc., Estados Unidos 1987.
- [105] Yourdon, E.N.; Constantine, L.L. *Structured Design*. Yourdon Press, 1978

Anexos

A. Diseño de la Materia Principios de Programación

1. Planeación didáctica de grupos de control
2. Planeación didáctica del grupo experimental
3. Paquete de prácticas de laboratorio

B. Evaluación de Profesores

1. Formato de evaluación de profesores
2. Resultados de evaluación previo al experimento (ciclo 02-05)
3. Resultados de evaluación posterior al experimento (ciclo 03-05)

C. Prueba de Habilidades de Programación (HaDiCA)

1. Diseño de rubricas de evaluación
2. Solución de problemas
3. Formato para concentrar datos de la aplicación

Anexo A-1

Planeación didáctica del grupo de control



UNIVERSIDAD DEL VALLE DE MEXICO

DIRECCIÓN ACADÉMICA

PLANEACION DIDÁCTICA

DEPARTAMENTO ACADÉMICO	TECNOCIENCIA
LICENCIATURA O POSGRADO EN	INGENIERÍA INDUSTRIAL Y DE SISTEMAS, ING. MECANICA INDUSTRIAL, ING. EN MECATRONICA, ING. TELECOMUNICACIONES Y ELECTRONICA
ASIGNATURA	PRINCIPIOS DE PROGRAMACION
SERIACIÓN	
SEMESTRE O CUATRIMESTRE EN QUE SE IMPARTE	1o.
FECHA DE REALIZACIÓN	Julio 2005

HORAS CON DOCENTE	HORAS INDEPENDIENTES	TOTAL DE HORAS SEMANA	TOTAL DE HORAS SEMESTRE O CUATRIMESTRE	CREDITOS
3	3	6	45/45=90	5.6

OBJETIVO GENERAL

El estudiante aplicará los conocimientos básicos de programación en la solución de problemas que involucren procesamiento de datos mediante un programa de cómputo, desarrollado en algún lenguaje de programación en particular; con una actitud comprometida y responsable.

BIBLIOGRAFÍA BÁSICA

- 📖 Deitel, Harvey M., Cómo programar en C/C++, México, Pearson Education, 2003, 4a, 970-26-0254-8.
- 📖 PARSONS, June Jamrich, Conceptos de Computación, México, International Thomson, 2004, 6a, 970-686-281-1.
- 📖 Microsoft Office 2000: introducción, México, International Thomson, 2000, 970-686-000-2.

1. INTRODUCCIÓN A LA PROGRAMACIÓN

HRS. 18

OBJETIVO ESPECÍFICO POR UNIDAD:

El estudiante identificará los elementos básicos de todo lenguaje de programación, comparando las posibles diferencias que existan entre ellos en cuanto a estructura, sintaxis y procedimientos.

No. DE SESION	FECHA	HRS. TEMA Y/O SUBTEMA	TEMA Y SUBTEMAS	ESTRATEGIAS DE ENSEÑANZA	EXPERIENCIA DE APRENDIZAJE	ACCIONES PARA EL DESARROLLO DE SUBHABILIDADES*
1	22-27 08/2005 sem.4	3 aula 3 AAI	1. Introducción a la programación 1.1 Introducción 1.2 Definición de algoritmos 1.3 Técnicas para elaboración de algoritmos	<ul style="list-style-type: none"> El docente Facilitará conceptos, simbología y analizará flujogramas de muestra con el alumno. 	<ul style="list-style-type: none"> El alumno memorizará los diversos símbolos utilizados en los flujogramas. 	Responsabilidad: <ul style="list-style-type: none"> Puntualidad en la entrega de tareas y/o practicas Cumplir con las políticas del laboratorio de cómputo. Leer previamente los temas a discutir en clase. Realizar todas las actividades y ejercicios asignadas por el docente Elaboración de mapas conceptuales Entrega de propuesta de proyecto de la materia
2	29-3 08/2005, 09/2005 sem.5,1	3 aula 3 AAI	1.4 Representación de algoritmos: flujogramas y pseudocódigo	<ul style="list-style-type: none"> Manejo de algún software para Diagramas de flujo (opcional) El estudiante realizará flujogramas elementales 	<ul style="list-style-type: none"> El alumno analizará diagramas de flujo y programas en pseudocódigo resueltos, tomando en cuenta el enunciado del problema, la identificación de entradas y salidas, así como los cálculos necesarios para llegar a los resultados esperados. 	
3	5-10 09/2005 sem.2	3 aula 3 AAI	1.5 Definición y tipos de datos y variables 1.6 Tipos de expresiones e instrucciones 1.7 Estructura general de un programa. 1.8 Entrada/Salida 1.9 Técnicas de programación	<ul style="list-style-type: none"> El docente Facilitará las características principales del software para la ejecución de programas en lenguaje C. ABP 	<ul style="list-style-type: none"> El alumno podrá dibujar y ejecutar flujogramas con el software correspondiente para comprobar sus resultados. El alumno realizara las prácticas que le indique el docente. 	

OBJETIVO ESPECÍFICO POR UNIDAD:

El estudiante creará programas de cómputo básicos utilizando estructuras de control de flujo de datos en la solución de problemas sencillos que requieren procesamiento de información.

No. DE SESION	FECHA	HRS. TEMA Y/O SUBTEMA	TEMA Y SUBTEMAS	ESTRATEGIAS DE ENSEÑANZA	EXPERIENCIA DE APRENDIZAJE	ACCIONES PARA EL DESARROLLO DE SUBHABILIDADES*
4	12-17 09/2005 sem.3	3 aula 3 AAI	2. Estructuras de control de flujo 2.1 Bifurcaciones 2.1.1 If	<ul style="list-style-type: none"> El docente Facilitará conceptos, sintaxis del lenguaje C y analizará programas de muestra con el alumno. El alumno desarrollará programas en lenguaje C. Se tomarán flujogramas de los libros para que el alumno los transcriba al lenguaje El alumno desarrollará programas en lenguaje C desde el inicio (sin diagramas de flujo resueltos), bajo la supervisión del docente en el laboratorio 	<ul style="list-style-type: none"> El alumno memorizará la sintaxis empleada para las diversas estructuras de control. El alumno ejecutará programas en lenguaje C para entender como funcionan las estructuras de control. El alumno ejecutará en modo de depuración sus programas para entender paso a paso la secuencia de las instrucciones y la afectación de las variables. El alumno realizara las prácticas que le indique el docente. Elaborar un mapa conceptual que integre todos los temas de la unidad. 	<p>Responsabilidad:</p> <ul style="list-style-type: none"> Puntualidad en la entrega de tareas y/o practicas Cumplir con las políticas del laboratorio de cómputo. Leer previamente los temas a discutir en clase. Realizar todas las actividades y ejercicios asignadas por el docente Elaboración de mapas conceptuales Entrega del avance del proyecto
5	19-24 09/2005 sem.4	3 aula 3 AAI	2.1.2 Switch			
6	26-1º. 09/2005, 10/2005 sem.5	3 aula 3 AAI	1er. Examen parcial, recepción y revisión de trabajos, retroalimentación			
7	3-8 10/2005 sem.1	3 aula 3 AAI	2.2 Ciclos 2.2.1 Do 2.2.2 While			
8	10-15 10/2005 sem.2	3 aula 3 AAI	2.2.3 For			

3. FUNCIONES

HRS. 18.

OBJETIVO ESPECÍFICO POR UNIDAD:

El estudiante creará programas de cómputo de complejidad intermedia utilizando estructuras de control, insertadas en subprogramas reutilizables que comparten información entre sí para optimizar su funcionamiento, identificando en consecuencia los elementos de la programación estructurada.

No. DE SESION	FECHA	HRS. TEMA Y/O SUBTEMA	TEMA Y SUBTEMAS	ESTRATEGIAS DE ENSEÑANZA	EXPERIENCIA DE APRENDIZAJE	ACCIONES PARA EL DESARROLLO DE SUBHABILIDADES*
9	17-22 10/2005 sem.3	3 aula 3 AAI	3. Funciones 3.1 Conceptos de procedimiento y parámetros	<ul style="list-style-type: none"> El docente Facilitará conceptos, sintaxis del lenguaje C y analizará programas de muestra con el alumno. 	<ul style="list-style-type: none"> El alumno memorizará la sintaxis empleada para las diversas estructuras de control. 	Responsabilidad: <ul style="list-style-type: none"> Puntualidad en la entrega de tareas y/o practicas Cumplir con las políticas del laboratorio de cómputo. Leer previamente los temas a discutir en clase. Realizar todas las actividades y ejercicios asignadas por el docente Elaboración de mapas conceptuales Entrega del avance del proyecto
10	24-29 10/2005 sem.4	3 aula 3 AAI	3.2 Funciones 3.2.1 Propias del lenguaje 3.2.2 Del programador	<ul style="list-style-type: none"> El alumno desarrollará programas en lenguaje C. Se tomarán flujogramas de los libros para que el alumno los transcriba al lenguaje 	<ul style="list-style-type: none"> El alumno ejecutará programas en lenguaje C para entender como funcionan las estructuras de control. 	
11	31-5 10/2005, 11/2005 sem.5,1	3 aula 3 AAI	2º. Examen parcial, recepción y revisión de trabajos, retroalimentación	<ul style="list-style-type: none"> El alumno desarrollará programas en lenguaje C desde el inicio (sin diagramas de flujo resueltos), bajo la supervisión del docente en el laboratorio Mapas conceptuales que integren todos los conceptos de la unidad 	<ul style="list-style-type: none"> El alumno ejecutará en modo de depuración sus programas para entender paso a paso la secuencia de las instrucciones y la afectación de las variables. El alumno realizara las prácticas que le indique el docente. 	

4. ESTRUCTURAS DE DATOS ESTÁTICAS Y DINÁMICAS

HRS. 24 .

OBJETIVO ESPECÍFICO POR UNIDAD:

El estudiante creará programas de cómputo vinculados a estructuras de datos conocidas residentes en la memoria principal de la computadora y, dependiendo de la construcción de dicha estructura, aplicará los procedimientos requeridos para la interacción con la información contenida en ellas.

No. DE SESION	FECHA	HRS. TEMA Y/O SUBTEMA	TEMA Y SUBTEMAS	ESTRATEGIAS DE ENSEÑANZA	EXPERIENCIA DE APRENDIZAJE	ACCIONES PARA EL DESARROLLO DE SUBHABILIDADES*
12,13	7-19 11/2005 sem.2,3	6 aula 6 AAI	4. Estructuras de datos estáticas y dinámicas 4.1 Arreglos de datos	<ul style="list-style-type: none"> El docente Facilitará conceptos, sintaxis del lenguaje C y analizará programas de muestra con el alumno. El alumno desarrollará programas en lenguaje C. Se tomarán como base ejemplos resueltos más comunes. El alumno desarrollará programas en lenguaje C bajo la supervisión del docente en el laboratorio. 	<ul style="list-style-type: none"> El alumno memorizará la sintaxis empleada para las estructuras de datos. El alumno ejecutará programas en lenguaje C para entender como funcionan las estructuras de datos. El alumno ejecutará en modo de depuración sus programas para entender paso a paso la secuencia de las instrucciones y la afectación de las variables. El alumno realizará las prácticas que le indique el docente. 	Responsabilidad: <ul style="list-style-type: none"> Puntualidad en la entrega de tareas y/o practicas Cumplir con las políticas del laboratorio de cómputo. Leer previamente los temas a discutir en clase. Realizar todas las actividades y ejercicios asignadas por el docente Elaboración de mapas conceptuales Entrega del proyecto final
14	21-26 11/2005 sem.4	3 aula 3 AAI	4.2 Registros			
15	28-3 11/2005, 12/2005 sem.5,1	3 aula 3 AAI	4.3 Apuntadores			
16	5-10 12/2005 sem.2	3 aula 3 AAI	3er. Examen parcial, recepción y revisión de trabajos, retroalimentación			

NO. DE UNIDAD: _____ NO. SESIÓN	ESTRATEGIAS DE EVALUACION DE CONTENIDOS PROGRAMÁTICOS	ESTRATEGIAS DE EVALUACIÓN DE SUBHABILIDADES	ESCENARIOS ACADÉMICOS	RECURSOS ACADÉMICOS	BIBLIOGRAFIA (BÁSICA Y COMPLEMENTARIA)	
Unidad 1 Sesiones: 1 – 6	• Examen escrito 50%	<ul style="list-style-type: none"> Evaluación Diagnóstica Elaboración de trabajos que se evaluarán con rúbrica Participación activa en clase Evaluación del trabajo en equipo: al preguntar a un integrante al azar se comprobará si todos participaron en el desarrollo del trabajo Mapas conceptuales 50% 	Laboratorio de cómputo	<ul style="list-style-type: none"> Pizarrón Cañón Web Software para diagramas de flujo Compilador de lenguaje C (Visual C++ ver.6.0 Microsoft) 	<p>Básica</p> <p>BROOKSHEAR, J. GLENN Computer Science. An overview. Ed. Pearson. USA, 2002. 7ª Edición.</p> <p>STAUGAARD, JR. Structured and Object Oriented Problem Solving Using C++. Ed. Prentice Hall. USA, 2002. 3ª Edición.</p> <p>COLLOPY, DAVID A Modular Approach. Ed. Pear-son. USA, 2002. 2ª Edición.</p> <p>Complementaria</p> <p>CAIRO, OSVALDO.</p>	
Unidad 2 Sesiones: 7 – 10						Unidad 2 Sesiones: 11 – 16
Unidad 3 Sesiones: 17 – 20	• Examen 50%	<ul style="list-style-type: none"> Elaboración de trabajos que se evaluarán con rúbrica Participación activa en clase Evaluación del trabajo en equipo: al preguntar a un integrante al azar se 	Laboratorio de cómputo	<ul style="list-style-type: none"> Pizarrón Cañón Web Compilador de lenguaje C (Visual 		

<p>Unidad 4</p> <p>Sesiones: 23 – 30</p>		<p>comprobará si todos participaron en el desarrollo del trabajo</p> <ul style="list-style-type: none"> • 30% • Elaboración y entrega de proyecto final = 20% 		<p>C++ ver.6.0 Microsoft)</p>	<p>Metodología de la programación, tomo I Ed AlfaOmega</p> <p>CARRETERO, GARCÍA, FERNANDEZ, CALDERÓN.</p> <p>El lenguaje de programación C - Diseño e implementación de programas Prentice Hall</p>
----------------------------------------------	--	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--	-----------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Anexo A-2

Planeación didáctica del grupo experimental



UNIVERSIDAD DEL VALLE DE MEXICO

DIRECCIÓN ACADÉMICA

PLANEACION DIDÁCTICA

DEPARTAMENTO ACADÉMICO	TECNOCIENCIA
LICENCIATURA O POSGRADO EN	INGENIERÍA INDUSTRIAL Y DE SISTEMAS, ING. MECANICA INDUSTRIAL, ING. EN MECATRONICA, ING. TELECOMUNICACIONES Y ELECTRONICA
ASIGNATURA	PRINCIPIOS DE PROGRAMACIÓN
SERIACIÓN	
SEMESTRE O CUATRIMESTRE EN QUE SE IMPARTE	1o.
FECHA DE REALIZACIÓN	Julio 2005

HORAS CON DOCENTE	HORAS INDEPENDIENTES	TOTAL DE HORAS SEMANA	TOTAL DE HORAS SEMESTRE O CUATRIMESTRE	CREDITOS
3	3	6	45/45=90	5.6

OBJETIVO GENERAL

El estudiante aplicará los conocimientos básicos de programación en la solución de problemas que involucren procesamiento de datos mediante un programa de cómputo, desarrollado en algún lenguaje de programación en particular; con una actitud comprometida y responsable.

BIBLIOGRAFÍA BÁSICA

- 📖 Deitel, Harvey M., Cómo programar en C/C++, México, Pearson Education, 2003, 4a, 970-26-0254-8.
- 📖 PARSONS, June Jamrich, Conceptos de Computación, México, International Thomson, 2004, 6a, 970-686-281-1.
- 📖 Microsoft Office 2000: introducción, México, International Thomson, 2000, 970-686-000-2.

1. INTRODUCCIÓN A LA PROGRAMACIÓN

HRS. 18 .

OBJETIVO ESPECÍFICO POR UNIDAD:

El estudiante identificará los elementos básicos de todo lenguaje de programación, comparando las posibles diferencias que existan entre ellos en cuanto a estructura, sintaxis y procedimientos.

No. DE SESION	FECHA	HRS. TEMA Y/O SUBTEMA	TEMA Y SUBTEMAS	ESTRATEGIAS DE ENSEÑANZA	EXPERIENCIA DE APRENDIZAJE	ACCIONES PARA EL DESARROLLO DE SUBHABILIDADES*
1	22-27 08/2005 sem.4	3 aula 3 AAI	5. Introducción a la programación 5.1 Introducción 5.2 Definición de algoritmos 5.3 Técnicas para elaboración de algoritmos 5.4 Definición de habilidades y la programación como una habilidad (adicional)	<ul style="list-style-type: none"> El docente Facilitará conceptos, simbología y analizará flujogramas de muestra con el alumno. 	<ul style="list-style-type: none"> El alumno memorizará los diversos símbolos utilizados en los flujogramas. 	Responsabilidad: <ul style="list-style-type: none"> Puntualidad en la entrega de tareas y/o practicas Cumplir con las políticas del laboratorio de cómputo. Leer previamente los temas a discutir en clase. Realizar todas las actividades y ejercicios asignadas por el docente Elaboración de mapas conceptuales Entrega de propuesta de proyecto de la materia
2	29-3 08/2005, 09/2005 sem.5,1	3 aula 3 AAI	5.5 Representación de algoritmos: flujogramas y pseudocódigo	<ul style="list-style-type: none"> Manejo de algún software para Diagramas de flujo (opcional) 	<ul style="list-style-type: none"> El alumno analizará diagramas de flujo y programas en pseudocódigo resueltos, tomando en cuenta el enunciado del problema, la identificación de entradas y salidas, así como los cálculos necesarios para llegar a los resultados esperados. 	
3	5-10 09/2005 sem.2	3 aula 3 AAI	5.6 Definición y tipos de datos y variables 5.7 Tipos de expresiones e instrucciones 5.8 Estructura general de un programa. 5.9 Entrada/Salida 5.10 Técnicas de programación 5.11 Presentación de la Metodología de Microingeniería (adicional)	<ul style="list-style-type: none"> El estudiante realizará flujogramas elementales El docente Facilitará las características principales del software para la ejecución de programas en lenguaje C. ABP 	<ul style="list-style-type: none"> El alumno podrá dibujar y ejecutar flujogramas con el software correspondiente para comprobar sus resultados. El alumno realizara las prácticas que le indique el docente. 	

2. ESTRUCTURAS DE CONTROL

HRS. 30

OBJETIVO ESPECÍFICO POR UNIDAD: El estudiante creará programas de cómputo básicos utilizando estructuras de control de flujo de datos en la solución de problemas sencillos que requieren procesamiento de información.

No. DE SESION	FECHA	HRS.TEMA Y/O SUBTEMA	TEMA Y SUBTEMAS	ESTRATEGIAS DE ENSEÑANZA	EXPERIENCIA DE APRENDIZAJE	ACCIONES PARA EL DESARROLLO DE SUBHABILIDADES*
4	12-17 09/2005 sem.3	3 aula 3 AAI	6. Estructuras de control de flujo	<ul style="list-style-type: none"> El docente Facilitará conceptos, sintaxis del lenguaje C y analizará programas de muestra con el alumno. El alumno desarrollará programas en lenguaje C. Se tomarán flujogramas de los libros para que el alumno los transcriba al lenguaje El alumno desarrollará programas en lenguaje C desde el inicio (sin diagramas de flujo resueltos), bajo la supervisión del docente en el laboratorio 	<ul style="list-style-type: none"> El alumno memorizará la sintaxis empleada para las diversas estructuras de control. El alumno ejecutará programas en lenguaje C para entender como funcionan las estructuras de control. El alumno ejecutará en modo de depuración sus programas para entender paso a paso la secuencia de las instrucciones y la afectación de las variables. El alumno realizara las prácticas que le indique el docente. Elaborar un mapa conceptual que integre todos los temas de la unidad. 	<p>Responsabilidad:</p> <ul style="list-style-type: none"> Puntualidad en la entrega de tareas y/o practicas Cumplir con las políticas del laboratorio de cómputo. Leer previamente los temas a discutir en clase. Realizar todas las actividades y ejercicios asignadas por el docente Elaboración de mapas conceptuales Entrega del avance del proyecto
5	19-24 09/2005 sem.4	3 aula 3 AAI	7. Reglas de la lógica (heurísticas) (adicional)			
6	26-1º. 09/2005, 10/2005 sem.5	3 aula 3 AAI	8. 1a iteración diagramas de secuencia (adicional) 9. Elementos de análisis (adicional) 10. Elementos de diseño (adicional) 11. Elementos de Codificación (adicional) 12. Aplicación de Pruebas (adicional)			
7	3-8 10/2005 sem.1	3 aula 3 AAI	12.1 Bifurcaciones 12.2 2ª iteración (adicional) 12.2.1 If 12.2.2 Switch			
8	10-15 10/2005 sem.2	3 aula 3 AAI	12.3 Elementos de análisis (adicional) 12.4 Elementos de diseño (adicional) 12.5 Elementos de codificación (adicional) 12.6 Aplicaición de pruebas (adicional) 12.7 Liberación (adicional) 1er. Examen parcial, recepción y revisión de trabajos, retroalimentación 12.8 Ciclos 12.9 3a iteración (adicional) 12.9.1 Do 12.9.2 While 12.9.3 For 12.10 Elementos de análisis (adicional) 12.11 Elementos de diseño (adicional) 12.12 Elementos de codificación (adicional) 12.13 Elmentos de aplicación de pruebas (adicional)			

3. FUNCIONES

HRS. 18.

OBJETIVO ESPECÍFICO POR UNIDAD:

El estudiante creará programas de cómputo de complejidad intermedia utilizando estructuras de control, insertadas en subprogramas reutilizables que comparten información entre sí para optimizar su funcionamiento, identificando en consecuencia los elementos de la programación estructurada.

No. DE SESION	FECHA	HRS. TEMA Y/O SUBTEMA	TEMA Y SUBTEMAS	ESTRATEGIAS DE ENSEÑANZA	EXPERIENCIA DE APRENDIZAJE	ACCIONES PARA EL DESARROLLO DE SUBHABILIDADES*
9	17-22 10/2005 sem.3	3 aula 3 AAI	12.14 Elementos de liberación (adicional) 13. Funciones	<ul style="list-style-type: none"> El docente Facilitará conceptos, sintaxis del lenguaje C y analizará programas de muestra con el alumno. 	<ul style="list-style-type: none"> El alumno memorizará la sintaxis empleada para las diversas estructuras de control. 	Responsabilidad:
10	24-29 10/2005 sem.4	3 aula 3 AAI	13.1 Conceptos de procedimiento y parámetros 13.2 Ejercicios de habilidades (adicional)	<ul style="list-style-type: none"> El alumno desarrollará programas en lenguaje C. Se tomarán flujogramas de los libros para que el alumno los transcriba al lenguaje 	<ul style="list-style-type: none"> El alumno ejecutará programas en lenguaje C para entender como funcionan las estructuras de control. 	<ul style="list-style-type: none"> Puntualidad en la entrega de tareas y/o practicas Cumplir con las políticas del laboratorio de cómputo.
11	31-5 10/2005, 11/2005 sem.5,1	3 aula 3 AAI	13.3 Funciones 13.3.1 Propias del lenguaje 13.3.2 Del programador 2º. Examen parcial, recepción y revisión de trabajos, retroalimentación	<ul style="list-style-type: none"> El alumno desarrollará programas en lenguaje C desde el inicio (sin diagramas de flujo resueltos), bajo la supervisión del docente en el laboratorio Mapas conceptuales que integren todos los conceptos de la unidad 	<ul style="list-style-type: none"> El alumno ejecutará en modo de depuración sus programas para entender paso a paso la secuencia de las instrucciones y la afectación de las variables. El alumno realizara las prácticas que le indique el docente. 	<ul style="list-style-type: none"> Leer previamente los temas a discutir en clase. Realizar todas las actividades y ejercicios asignadas por el docente Elaboración de mapas conceptuales Entrega del avance del proyecto

4. ESTRUCTURAS DE DATOS ESTÁTICAS Y DINÁMICAS

HRS. 24 .

OBJETIVO ESPECÍFICO POR UNIDAD:

El estudiante creará programas de cómputo vinculados a estructuras de datos conocidas residentes en la memoria principal de la computadora y, dependiendo de la construcción de dicha estructura, aplicará los procedimientos requeridos para la interacción con la información contenida en ellas.

No. DE SESION	FECHA	<i>HRS.</i> <i>TEMA Y/O SUBTEMA</i>	<i>TEMA Y SUBTEMAS</i>	ESTRATEGIAS DE ENSEÑANZA	EXPERIENCIA DE APRENDIZAJE	ACCIONES PARA EL DESARROLLO DE SUBHABILIDADES*
12,13	7-19 11/2005 sem.2,3	6 aula 6 AAI	14. Estructuras de datos estáticas y dinámicas 14.1 Ejercicios de habilidadess (adicional)	<ul style="list-style-type: none"> El docente Facilitará conceptos, sintaxis del lenguaje C y analizará programas de muestra con el alumno. 	<ul style="list-style-type: none"> El alumno memorizará la sintaxis empleada para las estructuras de datos. El alumno ejecutará programas en lenguaje C para entender como funcionan las estructuras de datos. 	Responsabilidad:
14	21-26 11/2005 sem.4	3 aula 3 AAI	14.2 Arreglos de datos 14.3 Ejercicios de habilidadess (adicional)	<ul style="list-style-type: none"> El alumno desarrollará programas en lenguaje C. Se tomarán como base ejemplos resueltos más comunes. 	<ul style="list-style-type: none"> El alumno ejecutará en modo de depuración sus programas para entender paso a paso la secuencia de las instrucciones y la afectación de las variables. 	<ul style="list-style-type: none"> Puntualidad en la entrega de tareas y/o practicas Cumplir con las políticas del laboratorio de cómputo.
15	28-3 11/2005, 12/2005 sem.5,1	3 aula 3 AAI	14.4 Registros 14.5 Ejercicios de habilidadess (adicional)	<ul style="list-style-type: none"> El alumno desarrollará programas en lenguaje C bajo la supervisión del docente en el laboratorio. 	<ul style="list-style-type: none"> El alumno realizará las prácticas que le indique el docente. 	<ul style="list-style-type: none"> Leer previamente los temas a discutir en clase. Realizar todas las actividades y ejercicios asignadas por el docente Elaboración de mapas conceptuales Entrega del proyecto final
16	5-10 12/2005 sem.2	3 aula 3 AAI	14.6 Apuntadores 14.7 Ejercicios de habilidadess (adicional)			
			3er. Examen parcial, recepción y revisión de trabajos, retroalimentación			

NO. DE UNIDAD: _____ NO. SESIÓN	ESTRATEGIAS DE EVALUACION DE CONTENIDOS PROGRAMÁTICOS	ESTRATEGIAS DE EVALUACIÓN DE SUBHABILIDADES	ESCENARIOS ACADÉMICOS	RECURSOS ACADÉMICOS	BIBLIOGRAFIA (BÁSICA Y COMPLEMENTARIA)
Unidad 1 Sesiones: 1 – 6	• Examen escrito 50%	<ul style="list-style-type: none"> Evaluación Diagnóstica Elaboración de trabajos que se evaluarán con rúbrica Participación activa en clase Evaluación del trabajo en equipo: al preguntar a un integrante al azar se comprobará si todos participaron en el desarrollo del trabajo Mapas conceptuales 50% 	Laboratorio de cómputo	<ul style="list-style-type: none"> Pizarrón Cañón Web Software para diagramas de flujo Compilador de lenguaje C (Visual C++ ver.6.0 Microsoft) 	<p>Básica</p> <p>BROOKSHEAR, J. GLENN Computer Science. An overview. Ed. Pearson. USA, 2002. 7ª Edición.</p> <p>STAUGAARD, JR. Structured and Object Oriented Problem Solving Using C++. Ed. Prentice Hall. USA, 2002. 3ª Edición.</p> <p>COLLOPY, DAVID A Modular Approach. Ed. Pear-son. USA, 2002. 2ª Edición.</p> <p>Complementaria</p> <p>CAIRO, OSVALDO.</p>
Unidad 2 Sesiones: 7 – 10					
Unidad 3 Sesiones: 17 – 20	Unidad 3 Sesiones: 21 – 22	<ul style="list-style-type: none"> Elaboración de trabajos que se evaluarán con rúbrica Participación activa en clase Evaluación del trabajo en equipo: al preguntar a un integrante al azar se 	Laboratorio de cómputo	<ul style="list-style-type: none"> Pizarrón Cañón Web Compilador de lenguaje C (Visual 	

<p>Unidad 4</p> <p>Sesiones: 23 – 30</p>		<p>comprobará si todos participaron en el desarrollo del trabajo</p> <ul style="list-style-type: none"> • 30% • Elaboración y entrega de proyecto final = 20% 		<p>C++ ver.6.0 Microsoft)</p>	<p>Metodología de la programación, tomo I Ed AlfaOmega</p> <p>CARRETERO, GARCÍA, FERNANDEZ, CALDERÓN.</p> <p>El lenguaje de programación C - Diseño e implementación de programas Prentice Hall</p>
----------------------------------------------	--	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--	-----------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Anexo A-3

Paquete de prácticas de laboratorio

Paquete de Prácticas para Principios de Programación

El Hermano Menor

Lucia tiene un hermano menor, Andres, que está estudiando geometría. En su materia tiene que hacer muchos ejercicios para calcular el área y el perímetro de distintas figuras básicas, Entre ellas un rectángulo, un círculo y un triángulo. Cada vez que Andres termina un ejercicio va con Lucia para preguntarle si le salió bien el resultado, entonces ella resuelve también el ejercicio y le dice a Andres si le salió bien. Como son muchos ejercicios Lucia decidió hacer un programa en el que ella pueda ingresar los valores de cada ejercicio de Andres y ver si este obtuvo el resultado correcto.

El Deportivo

Miguel Trabaja en un club deportivo. Parte de su trabajo consiste en orientar a los miembros sobre las condiciones del clima y el tipo de deporte que podrían realizar dependiendo de éste.

Para facilitar su trabajo Miguel decidió hacer un tablero electrónico en el cual se anuncie la temperatura del día y el tipo de deporte que se recomienda. El tablero es controlado por un programa que hizo Miguel en el cual éste escribe cada hora la temperatura y aparece en el tablero la temperatura y el deporte recomendado. La gerencia le dio a Miguel la siguiente tabla para su tablero electrónico (nota: la temperatura está en grados Fahrenheit y debe publicarse en centígrados)

Deporte	Temperatura
Natación	> 85
Tenis	$70 < \text{temp} \leq 85$
Golf	$32 < \text{temp} \leq 70$
Esquí	$10 < \text{temp} \leq 32$
Damas chinas	≤ 10

El Tren

Maria trabaja en la taquilla de una estación de trenes que solo vende viajes redondos. El precio del boleto calcula tomando en cuenta el número de kilómetros que se van a recorrer, siendo el

precio \$60.00 por kilómetro. Este precio puede tener un descuento del 30% si el viaje de regreso se hace después de 7 días del viaje de ida, o si el recorrido supera los 800 kilómetros. Maria hizo un programa para que ayude a determinar el precio de cada boleto que vende.

El Vendedor

Carlos es agente de Ventas. En esta semana ya realizó cinco ventas y desea conocer cual es el total de su comisión. La empresa dónde trabaja le dio las siguientes condiciones: Si la venta es menor a \$1,000.00 se le otorga el 3% de comisión. Si la venta es entre \$1,000.00 y \$5,000.00 el vendedor recibe el 5% de la comisión, y si es más de \$5,000.00 recibe el 7%. Carlos escribió un programa que lo ayuda a calcular la comisión de cada venta, y acumularla para obtener el total.

La Farmacia

Lupita y su marido Martín han decidido poner una farmacia. Martin se encarga de surtir las recetas y Lupita cobra la cuenta en la caja. Su trabajo consiste en ingresar el precio de cada medicina, calcular el descuento de cada una, obtener la suma total de las medicinas sin descuento, la suma total de los descuentos, contar el total de productos vendidos, obtener el total a pagar, recibir el dinero del pago y calcular el cambio. Todo esto lo hace con ayuda de un programa que hicieron entre los dos.

Los Volados

Jesus está muy aburrido y decidió hacer un programa para jugar volados contra la computadora. En su juego la computadora lanza una moneda virtual y mientras está en el aire nos pregunta que pedimos (águila o sol), luego nos informa que salió y si ganamos o perdimos. Este juego lo repite indefinidamente hasta que le indicamos que ya queremos terminar y entonces nos informa cuantos volados se hicieron, cuantas veces gano la computadora y cuantos nosotros.

El Ajedrez

Cuenta la leyenda que un viejo rey quedó fascinado con el juego del ajedrez, mandó llamar al inventor del juego y le dijo que lo recompensaría por su gran invención. El creador, después de pensarlo por un instante, le pidió al rey como recompensa que en el primer cuadro del tablero le diera un grano de trigo, en el segundo le diera dos, en el tercer cuadro le diera cuatro, y así sucesivamente el doble en cada cuadro. El rey pensó que eso no era mucho y lo concedió.

¿Puedes hacer un programa que calcule cuantos granos le correspondían al último cuadro y cuantos fueron en total?

La Inversionista

Verónica ha decidido comenzar a ahorrar para su vejez. Después de hablar con el gerente de un banco le informaron que cada mes las tasas de interés varían pero que podría reinvertir en cada ocasión su ahorro más el capital. Verónica hizo un programa en el cual está considerando ahorrar \$1,000 mensuales, si la tasa fuera la misma y ahorrara durante 25 años, ¿Cuánto dinero tendría al final de ahorro, de intereses y total?

La Aficionada

Durante navidad Mónica participó en un concurso de cantante aficionada que duró 12 días con la canción “The 12 Days of Christmas” (Los 12 días de navidad), durante el evento le dieron regalos en la siguiente forma: el primer día recibió una perdiz; el segundo día dos tórtolas y una perdiz; el tercer día tres gallinas de Guinea, dos tórtolas y una perdiz. Esto continuó durante los doce días. En el duodécimo día ella recibió $12 + 11 + \dots + 2 + 1$ regalos. ¿Cuántos obsequios fueron en conjunto?

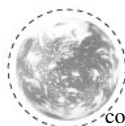
Anexo B-1

Formato de evaluación de profesores

**DIRECCION ACADEMICA
SISTEMA INTEGRAL DE EVALUACIÓN EDUCATIVA
EVALUACIÓN DE PARES
LICENCIATURA CUATRIMESTRAL
PERIODO 03 / 06**

CARRERA: _____
ACADEMIA: _____
FECHA: _____
HORARIO: _____

NOMBRE DEL FACILITADOR: _____
MATERIA: _____ CUATRIMESTRE: _____ GRUPO: _____



Instrucciones: Estimado facilitador el presente instrumento tiene como objetivo conocer tu apreciación acerca del desempeño de tu compañero de academia, principalmente en los aspectos relacionados al **manejo de los contenidos**; lo cual al realizar la retroinformación le permitirá apreciar sus fortalezas y atender sus áreas de oportunidad. Para tal efecto te solicitamos tu honesta y objetiva colaboración al momento de emitir tu opinión, marcando con una **X** el espacio que corresponda, de acuerdo a lo siguiente:

1. Totalmente en desacuerdo
2. En desacuerdo
3. Indeciso
4. De acuerdo
5. Totalmente de acuerdo

*	PUNTUACIÓN	1	2	3	4	5
	PLANEACIÓN					
	El facilitador					
1	Inició y terminó en el tiempo establecido su sesión de aprendizaje					
2	Revisó los contenidos de forma ordenada, durante la sesión de aprendizaje					
3	Brindó el tiempo suficiente para la realización de las actividades asignadas					
4	Llevó un orden adecuado y lógico durante el desarrollo de su sesión de aprendizaje					
	TOTAL					
	PROMEDIO					
	EJECUCIÓN	1	2	3	4	5
	El facilitador					
5	Utilizó fuentes de información actualizadas (publicaciones, internet, libros, enciclopedias, etc.) durante su sesión de aprendizaje					
6	Utilizó material didáctico apropiado para la revisión y comprensión del tema					
7	Llevó al análisis del tema, al grupo, durante la sesión de aprendizaje					
8	Llevó al grupo a deducir conclusiones sobre el tema					
9	Llevó a los alumnos a revisar el tema en un nivel cognitivo adecuado					
10	Promovió la participación activa del alumno, durante la sesión de aprendizaje					
11	Llevó a cabo la dinámica de la sesión de aprendizaje con orden y organización					
12	Utilizó un lenguaje claro y entendible, durante la sesión de aprendizaje					
13	Promovió la deducción de conceptos y términos propios de la materia dentro de la sesión de aprendizaje					
14	Aplicó los contenidos de la materia al promover la solución de problemas reales					
	TOTAL					
	PROMEDIO					
	MEDIOS DIDÁCTICOS	1	2	3	4	5
	El facilitador					
15	Fue creativo en la elección y manejo de sus recursos didácticos					
16	Construyó un ambiente propicio para el trabajo (orden físico, respeto etc.)					
17	Promovió la realización de actividades en equipo					
	TOTAL					
	PROMEDIO					
	MÉTODOS DE ENSEÑANZA	1	2	3	4	5
	El facilitador					
18	Sugirió la realización de actividades con el objetivo de reforzar el tema revisado durante la sesión de aprendizaje					
	TOTAL					
	PROMEDIO					
	MOTIVACIÓN	1	2	3	4	5
19	Motivó al grupo a mejorar su desempeño.					

20	Colaboró con los alumnos durante la realización de sus actividades o exposición de clase					
	TOTAL					
	PROMEDIO					
	HABILIDADES PROFESIONALES	1	2	3	4	5
	El facilitador:					
21	Promovió la responsabilidad de los alumnos a través del cumplimiento, en tiempo y forma, de las actividades asignadas					
	TOTAL					
	PROMEDIO					
	HABILIDADES HUMANAS	1	2	3	4	5
22	Se dirigió a los alumnos con educación respeto y amabilidad, durante la sesión de aprendizaje					
	TOTAL					
	PROMEDIO					

Observaciones: _____

Promedio final _____ Nivel _____

Nombre y firma
Profesor Evaluado

Nombre y firma
Profesor Evaluador

¡EL VALORAR NUESTRA FORTALEZAS Y ATENDER NUESTRAS ÁREAS DE OPORTUNIDAD NOS PERMITIRÁ IR HACIA LA MEJORA CONTINUA!

DIRECCION ACADEMICA
SISTEMA INTEGRAL DE EVALUACIÓN EDUCATIVA
AUTOEVALUACIÓN
LICENCIATURA CUATRIMESTRAL
Periodo 02 / 06

CARRERA: _____
 ACADEMIA: _____
 FECHA: _____

NOMBRE DEL FACILITADOR: _____
 MATERIA: _____ CUATRIMESTRE: _____ GRUPO: _____

Instrucciones: Estimado Facilitador el presente instrumento tiene como objetivo conocer la auto apreciación que tienes acerca de tu desempeño, lo cual te permitirá apreciar tus fortalezas y atender tus áreas de oportunidad. Para tal efecto te solicitamos tu honesta y objetiva colaboración marcando con una **X** el espacio que corresponda, de acuerdo a lo siguiente:

- 1 Totalmente en desacuerdo
- 2 En desacuerdo
- 3 Indeciso
- 4 De acuerdo
- 5 Totalmente de acuerdo

*	PUNTUACIÓN	1	2	3	4	5
	PLANEACIÓN					
1	Doy a conocer a los alumnos la carpeta de la materia o el programa al inicio del curso					
2	Mantengo una secuencia lógica de los temas, de acuerdo a la carpeta de la materia o al programa					
3	Inicio y termino en el tiempo establecido las sesiones de aprendizaje					
4	Reviso los contenidos de forma ordenada, durante las sesiones de aprendizaje					
5	Brindo el tiempo suficiente para la realización de las actividades asignadas					
6	Llevo un orden adecuado y lógico durante el desarrollo de las sesiones de aprendizaje					
	TOTAL					
	PROMEDIO					
	EJECUCIÓN	1	2	3	4	5
7	Utilizo fuentes de información actualizadas, confiables y científicas (publicaciones, internet, libros, enciclopedias, etc.)					
8	Utilizo materiales didácticos apropiados para la revisión y comprensión de los temas					
9	Llevo al análisis de los temas, al grupo, durante las sesiones de aprendizaje					
10	Llevo al grupo a deducir conclusiones sobre cada uno de los temas					
11	Llevo a los alumnos a revisar los temas en un nivel cognitivo adecuado					
12	Promuevo la participación activa de los alumnos, durante las sesiones de aprendizaje					
13	Exijo a los alumnos calidad en la realización de las actividades y trabajos entregados					
14	Llevo a cabo la dinámica de las sesiones de aprendizaje con orden y organización					
15	Dedico tiempo, durante las sesiones de aprendizaje para aclarar dudas e inquietudes, respecto a los temas tratados					
16	Utilizo un lenguaje claro y entendible, durante las sesiones de aprendizaje					
17	Promuevo la deducción de conceptos y términos propios de la materia, durante las sesiones de aprendizaje					
18	Utilizo alguna estrategia para cerciorarme que los temas fueron comprendidos por todo el grupo					
19	Aplico los contenidos de la materia al promover la solución de problemas reales					
	TOTAL					
	PROMEDIO					
	MEDIOS DIDÁCTICOS	1	2	3	4	5
20	Soy creativo en la elección y manejo de los recursos didácticos					
21	Construyo un ambiente propicio para el trabajo (orden físico, respeto etc.) durante las sesiones de aprendizaje					
22	Promuevo la realización de actividades en equipo					
	TOTAL					
	PROMEDIO					
	MÉTODOS DE ENSEÑANZA	1	2	3	4	5
23	Verifico los resultados de las actividades de aprendizaje solicitadas en los tiempos previamente establecidos					
24	Promuevo la realización de actividades extra aula					
25	Sugiero la realización de actividades con el objetivo de reforzar el tema revisado					

	durante las sesiones de aprendizaje					
	TOTAL					
	PROMEDIO					
	MOTIVACIÓN	1	2	3	4	5
26	Motivo al grupo a mejorar su desempeño.					
27	Colaboro con los alumnos durante la realización de sus actividades o exposición de clase					
28	Estimulo a los alumnos a la superación personal como resultado de su esfuerzo continuo					
	TOTAL					
	PROMEDIO					
	EVALUACIÓN	1	2	3	4	5
29	Establezco oportunamente con el grupo, los criterios de aceptación para cada una de las actividades					
30	Respeto los criterios de aceptación establecidos previamente con el grupo					
31	Considero todas las actividades de aprendizaje dentro de los criterios de aceptación					
32	Informo oportunamente a los alumnos sobre los avances, aciertos y errores de su desempeño					
33	Soy claro y específico en los criterios de aceptación para cada una de las actividades					
	TOTAL					
	PROMEDIO					
	HABILIDADES PROFESIONALES	1	2	3	4	5
34	Promuevo la responsabilidad de los alumnos a través del cumplimiento, en tiempo y forma, de las actividades asignadas					
	TOTAL					
	PROMEDIO					
	HABILIDADES HUMANAS	1	2	3	4	5
35	Genero una relación de respeto entre los alumnos					
36	Establezco un clima de confianza que le permite a los alumnos expresar sus ideas e inquietudes					
37	Promuevo el respeto a las ideas de otros miembros del grupo					
38	Me dirijo a los alumnos con educación respeto y amabilidad, durante las sesiones de aprendizaje					
39	Muestro tolerancia ante los problemas que surgen en el grupo					
40	Muestro en todo momento una actitud positiva ante el grupo					
	TOTAL					
	PROMEDIO					

Observaciones:

Promedio final _____

Nivel _____

Nombre y firma del Facilitador

¡EL VALORAR NUESTRA FORTALEZAS Y ATENDER NUESTRAS ÁREAS DE OPORTUNIDAD NOS PERMITIRA IR HACIA LA MEJORA CONTINUA!

Anexo B-2

Resultados de la evaluación previa al experimento

(Ciclo 02-05)

SISTEMA INTEGRAL DE EVALUACIÓN EDUCATIVA

Seguimiento Docente
Licenciatura 02/05

TECNOCIENCIAS

NOMBRE DEL CATEDRÁTICO	ACADEMIA ADMINISTRATIVA 25%	TRABAJO DE ACADEMIA 25%	SEGUIMIENTO DOCENTE 30%	CAPACITACIÓN Y ACTUALIZACIÓN 20%	TOTAL 100%
ALFARO CISNEROS MARTHA VELIA	17.00%	5.00%	28.75%	20.00%	70.75%
ANTONIO BERNAL REFUGIO	15.00%	14.00%	27.38%	0.00%	56.38%
BALTÁR RODRÍGUEZ ROSARIO	23.00%	14.00%	12.75%	0.00%	49.75%
BENÍTEZ ALVA SALVADOR	19.90%	24.00%	26.50%	0.00%	70.40%
CABRERA SÁNCHEZ MARIA ANTONIA	22.00%	22.00%	25.25%	6.60%	75.85%
CARRANZA MONTAÑO CARLOS	25.00%	25.00%	29.63%	20.00%	99.63%
CARRILLO GARZÓN LORENZO MARTÍN	19.00%	25.00%	27.00%	13.10%	84.10%
COLÍN LIMA RAÚL	19.00%	21.00%	27.25%	13.10%	80.35%
CONTRERAS HERNÁNDEZ SALVADOR	23.00%	25.00%	26.25%	0.00%	74.25%
CORTES CONTRERAS MARCO ANTONIO	17.00%	24.00%	14.25%	13.10%	68.35%
CRUZ GÓMEZ LUÍS MANUEL	25.00%	14.00%	28.00%	20.00%	87.00%
CRUZ LÓPEZ VERÓNICA PATRICIA	25.00%	23.00%	27.00%	20.00%	95.00%
CU LARA SILVIA KARINA	22.90%	25.00%	27.00%	13.10%	88.00%
DE LA O CISNEROS JOSÉ LUÍS	16.70%	20.00%	24.25%	0.00%	60.95%
DELGADO ACOSTA RODOLFO	9.00%	0.00%	28.75%	20.00%	57.75%
DÍAZ BAUTISTA ANA MA.	24.80%	25.00%	27.75%	20.00%	97.55%
DÍAZ SALINAS JOSÉ LUÍS	25.00%	24.00%	27.38%	20.00%	96.38%
DURAN AVILES CLAUDIA	23.00%	17.00%	14.25%	6.60%	60.85%
FELIX GONZÁLEZ NAZARIO	25.00%	25.00%	25.50%	6.60%	82.10%
GARCÍA DE LUNA BEATRIZ	25.00%	25.00%	27.25%	0.00%	77.25%
GARCÍA JARA DIEGO	23.00%	25.00%	25.50%	13.10%	86.60%
GÓMEZ AGUILAR RICARDO	11.50%	0.00%	23.25%	0.00%	34.75%
GONZÁLEZ GONZÁLEZ JUANA NAYELY	15.00%	13.00%	13.50%	0.00%	41.50%
GONZÁLEZ MARTÍNEZ JOSÉ JOEL	25.00%	0.00%	27.38%	0.00%	52.38%
GONZÁLEZ PANTOJA JOAQUÍN	18.90%	0.00%	11.25%	0.00%	30.15%
HAM FLORES CARLOS GUILLERMO	23.00%	25.00%	28.88%	20.00%	96.88%
HERNÁNDEZ CEDILLO GENARO	25.00%	23.00%	25.25%	20.00%	93.25%
HERNÁNDEZ PONCE DAVID	24.80%	25.00%	27.00%	20.00%	96.80%
JUSTINIANO OSEGUEDA LORENA GUADALUPE	20.00%	0.00%	28.25%	20.00%	68.25%
LEÓN GÓMEZ JOSÉ LUÍS	16.20%	17.00%	17.50%	0.00%	50.70%
LEYVA MENDOZA JOSÉ SACRAMENTO GUADALUPE	14.90%	23.00%	12.00%	20.00%	69.90%
LLAMAS TORRES ENRIQUE	20.80%	22.00%	28.00%	20.00%	90.80%
LÓPEZ VEGA MIGUEL ÁNGEL	23.80%	25.00%	0.00%	20.00%	68.80%
MANZANO GARCÍA EDUARDO	18.30%	15.00%	16.50%	6.60%	56.40%
MÁRQUEZ VEGA JUAN	12.50%	0.00%	30.00%	20.00%	62.50%
MARTÍNEZ DE LA TORRE MARIA GUADALUPE	25.00%	25.00%	28.50%	20.00%	98.50%
MARTÍNEZ VÁZQUEZ JOSÉ GERARDO FILEMON	25.00%	0.00%	0.00%	6.60%	31.60%
MATUZ PARRA JUAN	24.00%	25.00%	29.75%	20.00%	98.75%
MEDINA SANTIAGO ALEJANDRO	23.00%	25.00%	26.00%	13.10%	87.10%
MENDOZA MALDONADO JOSÉ LUÍS	24.00%	15.00%	28.50%	6.60%	74.10%
MONTIEL RENTARÍA CARLOS	11.00%	11.00%	12.75%	13.10%	47.85%

NOMBRE DEL CATEDRÁTICO	ACADEMIA ADMINISTRATIVA 25%	TRABAJO DE ACADEMIA 25%	SEGUIMIENTO DOCENTE 30%	CAPACITACIÓN Y ACTUALIZACIÓN 20%	TOTAL 100%
MORALES LÓPEZ JULIO CESAR	25.00%	25.00%	28.13%	20.00%	98.13%
ORDÓÑEZ MONDRAGÓN HERIBERTO	23.00%	23.00%	27.00%	13.10%	86.10%
ORTIZ REYES CLAUDIA ISABEL	13.00%	0.00%	0.00%	6.60%	19.60%
PINEDO CHAVARIN JOSÉ	20.70%	22.00%	25.13%	13.10%	80.93%
RAMÍREZ HERRERA GABRIEL	14.00%	0.00%	26.25%	13.10%	53.35%
RESÉNDIZ FERNÁNDEZ GABRIEL	15.00%	25.00%	25.50%	20.00%	85.50%
REYES SÁNCHEZ JOSÉ ALFREDO	25.00%	20.00%	29.25%	0.00%	74.25%
RODRÍGUEZ FLORES VÍCTOR	11.60%	21.00%	0.00%	13.10%	45.70%
RODRÍGUEZ LÓPEZ FELIPE ARMANDO	17.90%	17.00%	28.50%	13.60%	77.00%
ROLDAN GONZÁLEZ JOSÉ DE JESÚS	24.70%	15.00%	26.00%	6.60%	72.30%
ROQUE MORENO MARIO	25.00%	25.00%	25.13%	20.00%	95.13%
SALDAÑA SÁNCHEZ SERGIO	25.00%	25.00%	22.88%	20.00%	92.88%
TRONCOZO BOCANEGRA VERÓNICA ALEJANDRA	22.90%	25.00%	28.13%	20.00%	96.03%
VALDEZ ALEMÁN EVA	24.00%	25.00%	25.88%	13.10%	87.98%
VARGAS DE BASTERRA RICARDO	25.00%	25.00%	13.88%	13.10%	76.98%
VARGAS GONZÁLEZ ADANELI	23.00%	0.00%	27.75%	6.60%	57.35%
VEGA OBREGÓN SILVIA	22.00%	22.00%	25.88%	20.00%	89.88%
VERGARA SÁNCHEZ JOSÉ ARMANDO	25.00%	22.00%	27.75%	13.10%	87.85%
ZAMORA CORTES JESÚS	23.00%	20.00%	28.00%	13.10%	84.10%

 Lic. Beatriz Elizabeth Aguilar Lara
 Depto. Desarrollo Docente y Evaluación Educativa

Anexo B-3

Resultados de la evaluación posterior al experimento

(Ciclo 03-05)

SISTEMA INTEGRAL DE EVALUACIÓN EDUCATIVA

Seguimiento Docente

Licenciatura 03/05

TECNOCIENCIAS

CLAVE	NOMBRE DEL DOCENTE	RESULTADO	GLOBAL	CATEGORÍA
16794	CRUZ LOPEZ VERONICA PATRICIA	71.23	B+	BUENO (+)
16805	VARGAS DE BASTERRA RICARDO	64.5	B	BUENO
16842	MORALES LOPEZ JULIO CESAR	72.9	E	EXCELENTE
16871	GONZALEZ GONZALEZ JUANA NAYELI	68	B+	BUENO (+)
16899	LEON GOMEZ JOSE LUIS	54.2	B-	BUENO (-)
16903	DIAZ SALINAS JOSE LUIS	60.55	B	BUENO
16918	BALTAR RODRIGUEZ MARIA DEL ROSARIO	70.6	B+	BUENO (+)
33061	MEDINA SANTIAGO ALEJANDRO	64.53	B	BUENO
33265	CARRANZA MONTAÑO CARLOS	60.67	B	BUENO
33279	MORA RODARTE OLGA	65.7	B+	BUENO (+)
33282	HERNANDEZ SANCHEZ LUIS	61.9	B	BUENO
33296	GARCIA MUÑOZ EDUARDO	68.9	B+	BUENO (+)
33302	TRONCOSO BOCANEGRA VERONICA ALEJANDRA	68.14	B+	BUENO (+)
33314	DIAZ BAUTISTA ANA MA.	66.04	B+	BUENO (+)
33319	HAM FLORES CARLOS GUILLERMO	61.45	B	BUENO
33327	RODRIGUEZ LOPEZ FELIPE ARMANDO	61.65	B	BUENO
33335	FLORES SANCHEZ DARIO	44.2	R+	REGULAR (+)
33339	ZAMORA CORTES JESUS	58.4	B	BUENO
33354	CARD MENDEZ EDUARDO MARIO	71.8	B+	BUENO (+)
33366	ROQUE MORENO MARIO	61.85	B	BUENO
33368	OROPEZA GODEN MARTHA	56.2	B-	BUENO (-)
33369	MARTINEZ DE LA TORRE MARIA GUADALUPE	67.03	B+	BUENO (+)
33371	CONTRERAS HERNANDEZ SALVADOR	64.2	B	BUENO
33380	GUTIERREZ GUTIERREZ MARISELA	66.85	B+	BUENO (+)
33394	MANZANO GARCIA EDUARDO	65.1	B	BUENO
33402	DE LA O CISNEROS JOSE LUIS	56.85	B-	BUENO (-)
33403	CORTES CONTRERAS MARCO ANTONIO	68.55	B+	BUENO (+)
33404	GARCIA JARA DIEGO	56.04	B-	BUENO (-)
33405	RESENDIZ FERNANDEZ GABRIEL	63.95	B	BUENO
33406	PINEDO CHAVARIN JOSE	63.73	B	BUENO
33408	LEYVA MENDOZA JOSE SACRAMENTO GUA	61.2	B	BUENO
33410	FELIX GONZALEZ NAZARIO	60.56	B	BUENO
33412	VERGARA SANCHEZ JOSE ARMANDO	61	B	BUENO
33418	RODRIGUEZ FLORES VICTOR	59.03	B	BUENO
33429	HERNANDEZ PONCE DAVID	59.5	B	BUENO
33448	GARCIA DE LUNA BEATRIZ	65.6	B+	BUENO (+)
33457	VEGA OBREGON SILVIA	61.3	B	BUENO
33462	VELAZQUEZ VILLALOBOS CLAUDIA LORENA	74.9	E	EXCELENTE
33465	ORDÓÑEZ MONDRAGON HERIBERTO	61.53	B	BUENO
33481	BENITEZ ALVA SALVADOR	72.25	B+	BUENO (+)
33482	LLAMAS TORRES ENRIQUE	69.98	B+	BUENO (+)
33483	HERNANDEZ CEDILLO GENARO	48.95	R+	REGULAR

				(+)
33496	CU LARA SILVIA CARINA	68.6	B+	BUENO (+)
33498	DURAN AVILES CLAUDIA	67.77	B+	BUENO (+)
33501	VALDEZ ALEMAN EVA	71.03	B+	BUENO (+)
33512	ORTIZ REYES CLAUDIA ISABEL	60.8	B	BUENO
33518	CRUZ GOMEZ LUIS MANUEL	68.08	B+	BUENO (+)
33519	CARRILLO GARZON LORENZO MARTIN	58	B-	BUENO (-)
33522	REYES SANCHEZ JOSE ALFREDO	67.82	B+	BUENO (+)
33524	COLIN LIMA RAUL	59.9	B	BUENO
33528	CABRERA SANCHEZ MARIA ANTONIA	68.65	B+	BUENO (+)
33529	JUAREZ VALENCIA ARMANDO ENRIQUE	67.57	B+	BUENO (+)
33532	VALDES CASTRO YADIRA ROXANA	64.2	B	BUENO
33544	JIMENEZ LOZANO PILAR PATRICIA	62.13	B	BUENO
33565	ROLDAN GONZALEZ JOSE DE JESUS	68.9	B+	BUENO (+)
33566	SALDAÑA SANCHEZ SERGIO	68.3	B+	BUENO (+)
33569	MENDOZA MALDONADO JOSE LUIS	66.4	B+	BUENO (+)
33581	MARTINEZ VAZQUEZ JOSE GERARDO FILEMO	48.4	R+	REGULAR (+)
33605	GONZALEZ PANTOJA JOSE JOAQUIN	60.4	B	BUENO

 Lic. Beatriz Elizabeth Aguilar Lara
 Depto. Desarrollo Docente y Evaluación Educativa

Anexo C-1
Hoja de problemas

PRUEBA DE HABILIDADES DE DISEÑO Y CODIFICACIÓN DE ALGORITMOS

Realizar el Análisis, Diseño y Codificación de al menos dos de los siguientes ejercicios:

Problema 1. El jardín

Pedro es jardinero y le han pedido hacer un presupuesto para poner pasto en una casa que se está construyendo. El jardín es rectangular pero tiene una gran fuente circular en el centro. Pedro tiene que decirle a su cliente cuantos metros cuadrados de pasto tendrá que colocar y el importe total. Además Pedro cobra una cantidad por colocación que corresponde a su mano de obra. Alma es sobrina de Pedro y le dijo que podía hacerle un programa para que le ayudara a calcular lo que le piden.

Problema 2. El cuarto del bebe

Juan y Martha se acaban de mudar a su nueva casa. Ellos tienen un pequeño bebe de tan solo 2 años de edad y quieren redecorar su habitación. Martha está preocupada porque les alcance el dinero y Juan quiere tenerlo listo antes del cumpleaños número tres del bebe que será dentro de dos semanas. Después de conversar un rato Juan decide hacer un programa de cómputo para saber cuanto cuesta y cuanto tiempo se requerirá para pintar la habitación y su techo. Dos de las paredes cuentan con dos ventanas cada una y una de ellas tiene una puerta de acceso.

Problema 3. El viaje

Angelica es estudiante de Sistemas y quiere irse de intercambio a Canadá durante un cuatrimestre. Sabe que durante esos cuatro meses tendrá que cubrir su boleto de ida y vuelta, su colegiatura, su hospedaje y sus alimentos; además, quiere disponer de una cantidad semanal para diversiones todo esto en dólares canadienses. Durante el curso hay una semana de receso en la que quiere hacer un viaje a Nueva York para conocerlo y las agencias le proponen un paquete todo pagado en dólares americanos. Angelica quiere hacer un programa para calcular en pesos cuanto dinero necesita para su intercambio.

Problema 4. La escuela

Andrés está haciendo su servicio social en una pequeña escuela primaria como asistente de la coordinación académica y le han pedido que obtenga una estadística para saber cual es el promedio global de calificaciones de los 25 alumnos de 6° año de primaria así como cuantos alumnos están reprobados, cuantos tienen un promedio entre 6 y 7.9 y cuantos tienen promedio arriba de 8. Cada estudiante lleva doce materias. Andrés pensó que es mejor hacer un programa que lo ayude a obtener esa estadística que hacer los cálculos a mano.

Anexo C-2

Diseño de rúbricas de evaluación

TABLA RUBRICA DE ANÁLISIS

CONCEPTO	NULO	BAJO	ALTO	COMPLETO
RESULTADOS	No hay identificación de Resultados	Hay poca identificación de Resultados	Hay suficiente identificación de Resultados	Están identificados todos los Resultados
Puntaje	1	2	3	4
PROCESOS	No hay identificación de Procesos	Hay poca identificación de Procesos	Hay suficiente identificación de Procesos	Están identificados todos los Procesos
Puntaje	1	2	3	4
ENTRADAS NECESARIAS	No hay identificación de las Entradas Necesarias	Hay poca identificación de las Entradas Necesarias	Hay suficiente identificación de las Entradas Necesarias	Están identificadas todas las Entradas Necesarias
Puntaje	1	2	3	4
RESTRICCIÓNES/CONDICIONES	No hay identificación de Restricciones o Condiciones	Hay poca identificación de Restricciones o Condiciones	Hay suficiente identificación de Restricciones o Condiciones	Están identificadas todas las Restricciones o Condiciones
Puntaje	1	2	3	4
ITERACIONES/REPETICIONES	No hay identificación de Iteraciones o Repeticiones	Hay poca identificación de Iteraciones o Repeticiones	Hay suficiente identificación de Iteraciones o Repeticiones	Están identificadas todas las Iteraciones o Repeticiones
Puntaje	1	2	3	4
DICCIONARIO DE DATOS	No hay identificación de los datos a utilizar y sus tipos	Hay poca identificación de los datos a utilizar y sus tipos	Hay suficiente identificación de los datos a utilizar y sus tipos	Están identificados todos los datos a utilizar y sus tipos
Puntaje	1	2	3	4

Puntaje Mínimo en Tabla de Análisis: 6
 Puntaje Máximo en Tabla de Análisis: 24

Definiciones:

Habilidad de Análisis.- Capacidad de identificar todos los componentes de un problema para poder proponer una solución. Está conformada por seis descriptores o componentes.

Identificación de Resultados (Cuantitativo).- Corresponde al proceso de identificar cuales son las salidas que requiere la solución del problema. Se mide precisamente en el número de salidas identificadas.

Identificación de Procesos (Cuantitativo).- Corresponde a los cálculos que requiere la solución del problema. Se mide en función del número de formulas identificadas para obtener los resultados requeridos. El número puede variar dependiendo del diseño de dichas formulas.

Identificación de Entradas (Cuantitativo).- Equivale al número de datos que son requeridos por las formulas identificadas para poder obtener los resultados esperados. El número puede variar dependiendo del diseño de las fórmulas o procesos.

Identificación de Condiciones o Restricciones (Cuantitativo).- Se consideran dos tipos de condiciones: Las que determinan el dominio de los datos de entrada por efecto de validaciones y aquellas que aplican sobre resultados intermedios o finales para decidir sobre acciones a tomar por necesidades planteadas por el problema.

Identificación de iteraciones o repeticiones (Cuantitativo).- Corresponde a las acciones de lectura, de cálculo o de salida que deben ser repetidas. Se mide en el número de acciones a repetir identificadas para cada uno de los tipos citados.

Diccionario de Datos (Cualitativo).- Es el concentrado de datos a utilizar por el problema, considera tanto datos de entrada, resultados parciales y los resultados de salida.

TABLA RUBRICA DE DISEÑO

CONCEPTO	NULO	BAJO	ALTO	COMPLETO
SECUENCIAS	No hay diseño claro de Secuencias	Hay un pobre diseño de Secuencias	Hay suficiente diseño de Secuencias	Están completamente diseñadas todas las Secuencias
Puntaje	1	2	3	4
CONDICIONES	No hay diseño claro de de Condiciones	Hay un pobre diseño de Condiciones	Hay suficiente diseño de Condiciones	Están completamente diseñadas todas las Condiciones
Puntaje	1	2	3	4
ITERACIONES	No hay diseño claro de Iteraciones o Repeticiones	Hay un pobre diseño de Iteraciones o Repeticiones	Hay suficiente diseño de Iteraciones o Repeticiones	Están completamente diseñadas todas las Iteraciones o Repeticiones
Puntaje	1	2	3	4
PRUEBAS	No hay diseño claro de Pruebas	Hay un pobre diseño de Pruebas	Hay suficiente diseño de Pruebas	Están completamente diseñadas todas las Pruebas
Puntaje	1	2	3	4
INTEGRACIÓN	No se aprecia en el diseño la Integración de las estructuras	Hay poca Integración en el diseño de las estructuras	Hay suficiente Integración en el diseño de las estructuras	En el diseño se aprecian bien Integradas todas las estructuras
Puntaje	1	2	3	4
CONSISTENCIA	No se aprecia relación alguna entre el análisis y el diseño	Hay poca relación entre el análisis y el diseño	Hay suficiente relación entre el análisis y el diseño	Se aprecia buena relación entre el análisis y el diseño
Puntaje	1	2	3	4

Puntaje Mínimo en Tabla de Análisis: 6

Puntaje Máximo en Tabla de Análisis: 24

Definiciones:

Habilidad de Diseño.- Capacidad de modelar una secuencia ordenada y lógica de pasos que a partir de los datos de entrada pueda producir los resultados señalados en el análisis considerando las condiciones e iteraciones identificadas. El modelo de solución debe respetar y aplicar las estructuras definidas por el Teorema de Jacopini. Esta habilidad está compuesta por seis descriptores o componentes.

Diseño de Secuencias (Cuantitativo).- Cada proceso identificado en el análisis requiere de datos para poder realizar su cálculo y genera resultados. Las secuencias deben estar ordenadas de tal forma que en conjunto puedan producir correctamente las salidas. Se mide a través del número de procesos correctamente ordenados.

TABLA RUBRICA DE CODIFICACIÓN

CONCEPTO	NULO	BAJO	ALTO	COMPLETO
FUNCIONALIDAD	No resuelve la tarea para la que fue creado	Resuelve pocas cosas de la tarea para la que fue creado	Resuelve casi toda la tarea para la que fue creado	Resuelve completamente toda la tarea para la que fue creado
Puntaje	1	2	3	4
CONFIABILIDAD	No tiene ningún mecanismo de validación o recuperación	Tiene pocos mecanismos de validación o recuperación	Tiene varios mecanismos de validación y recuperación	Tiene todos los mecanismos de validación y recuperación
Puntaje	1	2	3	4
USABILIDAD	No es comprensible su uso para el usuario	Es posible su uso pero su comunicación con el usuario es pobre	Es posible utilizarlo y su comunicación con el usuario es suficiente	Su uso es comprensible y tiene buena comunicación con el usuario
Puntaje	1	2	3	4
EFICIENCIA	No hay intención alguna de mejorar la eficiencia en tiempo de ejecución o uso de recursos	Hay poca intención de mejorar la eficiencia en tiempo de ejecución o uso de recursos	Hay intención de mejorar la eficiencia en tiempo de ejecución y uso de recursos	El algoritmo está optimizado en tiempo de ejecución y uso de recursos
Puntaje	1	2	3	4
MANTENIBILIDAD	El código no es claro ni comprensible	El código es poco claro y poco comprensible	El código es suficientemente claro y comprensible	El código es completamente claro y comprensible
Puntaje	1	2	3	4
PORTABILIDAD	NO APLICABLE	NO APLICABLE	NO APLICABLE	NO APLICABLE
Puntaje	0	0	0	0
CONSISTENCIA	No se aprecia relación alguna entre el diseño y la construcción	Hay poca relación entre el diseño y la construcción	Hay suficiente relación entre el diseño y la construcción	Se aprecia completa relación entre el diseño y la construcción
Puntaje	1	2	3	4

Puntaje Mínimo en Tabla de Codificación: 6

Puntaje Máximo en Tabla de Análisis: 24

Anexo C-3
Solución de problemas

SOLUCIÓN A LOS PROBLEMAS DE HADICA

Problema 1: El Jardín

Pedro es jardinero y le han pedido hacer un presupuesto para poner pasto en una casa que se está construyendo. El jardín es rectangular pero tiene una gran fuente circular en el centro. Pedro tiene que decirle a su cliente cuantos metros cuadrados de pasto tendrá que colocar y el importe total. Además Pedro cobra una cantidad por colocación que corresponde a su mano de obra. Alma es sobrina de Pedro y le dijo que podía hacerle un programa para que le ayudara a calcular lo que le piden.

Análisis

+ Identificación de Resultados

Presupuesto para poner pasto en un jardín
Total de metros cuadrados a Sembrar

+ Identificación de Procesos

Presupuesto = costoTotalManoObra + costoTotalPasto

costoTotalManoObra = tmc * mo

costoTotalPasto = tmc * costoPasto

tmc = areaRectangulo – areaCirculo

areaRectangulo = base * altura

areaCirculo = 3.1416 * radio * radio

+ Identificación de Datos de Entrada

Costo de la Mano de Obra por metro cuadrado de colocación de pasto.

Costo del metro cuadrado de pasto

Lado más largo del terreno donde se sembrará el pasto (base)

Lado más corto del terreno donde se sembrará el pasto (lado)

Radio del círculo que forma la fuente que se colocará al centro del terreno.

+ Identificación de restricciones o condiciones

La mano de obra debe ser mayor a 0

El lado más largo (base) del rectángulo del terreno debe ser mayor a 0

El lado más corto (altura) del rectángulo del terreno debe ser mayor a 0

El radio de la fuente debe ser mayor a 0

El área del rectángulo tiene que ser mayor que el área de la fuente.

+ Repeticiones o iteraciones

Repetir la lectura de los valores de las variables de entrada en caso de que sean incorrectas

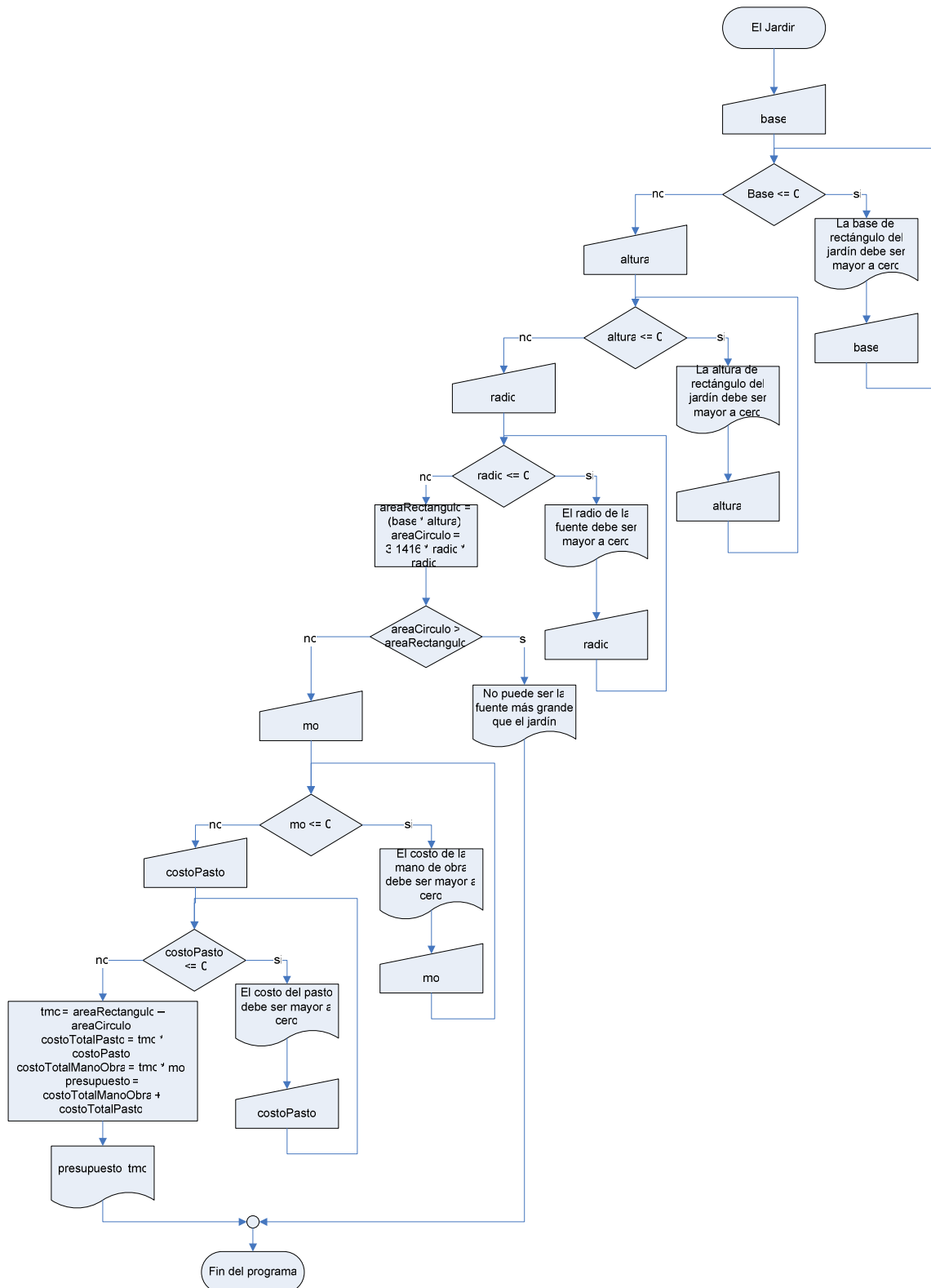
+ Diccionario de Datos

Nombre	Descripción	Tipo	Categoría	Dominio
presupuesto	Presupuesto total de sembrar pasto en el jardín	Real	Salida	Presupuesto > 0
tmc	Total de metros cuadrados de pasto requeridos	Real	Salida	tmc > 0
mo	Mano de Obra de colocación por metro	Real	Entrada	mo > 0
areaRectangulo	Total de metros cuadrados del rectángulo en donde se sembrará el pasto	Real	Proceso	areaRectangulo > 0
areaCirculo	Total de metros cuadrados	Real	Proceso	$0 < \text{areaCirculo} < \text{areaRectangulo}$
costoTotalManoObra	Costo total de la mano de obra necesaria	Real	Proceso	costoTotalManoObra > 0
costoTotalPasto	Costo total de los metros cuadrados de pasto a comprar	Real	Proceso	costoTotalPasto > 0
base	Lado más largo del rectángulo del terreno dónde se sembrará el jardín	Real	Entrada	base > 0
altura	Lado más corto del terreno dónde se sembrará el jardín	Real	Entrada	altura > 0
radio	Radio de la fuente central	Real	Entrada	radio > 0
costoPasto	Costo del metro cuadrado de pasto	Real	Entrada	costoPasto > 0

Métricas de Análisis:

CONCEPTO	CANTIDAD	OBSERVACIONES
Resultados a obtener	2	
Procesos a realizar	6	Podría variar el número dependiendo de la fórmula diseñada para el cálculo. El mínimo serían dos.
Datos de entrada requeridos	5	
Condiciones por reglas de negocio	1	
Condiciones por validaciones	5	Estas condiciones se pueden implementar a través de ciclos abiertos
Repeticiones por validaciones	5	Corresponden a las condiciones encontradas para los datos entrada
Repeticiones por reglas de negocio	0	
Diccionario de Datos	11	

Diseño



Métricas Cuantitativas del Diseño:

CONCEPTO	CANTIDAD	OBSERVACIONES
Diseño de Secuencias	2	
Diseño de Condiciones por validaciones	5	Dependiendo del diseño la validación puede aplicarse por condición o por iteración.
Diseño de Condiciones por reglas de negocio	1	
Diseño de Iteraciones por validaciones	5	Dependiendo del diseño la validación puede aplicarse por condición o por iteración.
Diseño de iteraciones por reglas de negocio	0	

Métricas Cualitativas del Diseño:

CONCEPTO	DESCRIPCIÓN
Pruebas	Es posible la ejecución de pruebas de caja negra y de caja blanca.
Integración	Se aplicaron correctamente las estructuras de control y las reglas de la lógica.
Consistencia	La relación entre el análisis y el diseño es consistente.

Construcción

```
using System;

namespace JardinPresupuesto
{
    /// <summary>
    /// Descripción breve de Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// Punto de entrada principal de la aplicación.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            double presupuesto, tmc, mo, areaRectangulo, areaCirculo,
                costoTotalManoObra, costoTotalPasto, lado, altura, radio,
                costoPasto;

            /// Poner la lectura de las variables x y y
            Console.Write("Dime la base: ");
            lado = System.Double.Parse(Console.ReadLine());
            while (lado<=0)
            {
                Console.WriteLine("La base del rectángulo debe ser mayor a
                    cero");
                Console.Write("Dime la base: ");
                lado = System.Double.Parse(Console.ReadLine());
            }
            Console.Write("Dime la altura: ");
            altura=System.Double.Parse(Console.ReadLine());
            while (altura <=0)
```



```
{
    Console.WriteLine("La altura del rectángulo debe ser mayor a
        cero");
    Console.Write("Dime la altura: ");
    altura=System.Double.Parse(Console.ReadLine());
}
Console.Write("Dime el valor de el radio: ");
radio = System.Double.Parse(Console.ReadLine());
while (radio<=0)
{
    Console.WriteLine("El radio de la fuente debe ser mayor a cero");
    Console.Write("Dime el valor de el radio: ");
    radio = System.Double.Parse(Console.ReadLine());
}
areaRectangulo=lado*altura;
areaCirculo=3.1416*radio*radio;
if (areaCirculo > areaRectangulo)
{
    Console.WriteLine("El área de la fuente no puede ser mayor que el
        jardin");
}
else
{
    Console.Write("Dime el costo de la mano de obra ");
    mo=System.Double.Parse(Console.ReadLine());
    while (mo<=0)
    {
        Console.WriteLine("El costo de la mano de obra debe ser mayor a
            cero");
        Console.Write("Dime el costo de la mano de obra ");
        mo=System.Double.Parse(Console.ReadLine());
    }
    Console.Write("Dime el costo del pasto ");
    costoPasto=System.Double.Parse(Console.ReadLine());
    while (costoPasto<=0)
    {
        Console.WriteLine("El costo del pasto debe ser mayor a cero");
        Console.Write("Dime el costo del pasto ");
        costoPasto=System.Double.Parse(Console.ReadLine());
    }
    tmc = areaRectangulo - areaCirculo;
    costoTotalPasto = tmc * costoPasto;
    costoTotalManoObra = tmc * mo;
    presupuesto = costoTotalManoObra + costoTotalPasto;
    Console.WriteLine("El total de Metros cuadrados de pasto es:
        {0}",tmc);
    Console.WriteLine("El presupuesto total es: {0}",presupuesto);
}
Console.ReadLine();
}
}
```

Métricas Cuantitativas de Construcción:

CONCEPTO	CANTIDAD
Número de lectura de variables validadas	5
Número de lectura de variables no validada.	0
Número de condiciones basadas en reglas del negocio	1
Número de condiciones por validaciones	0
Número de ciclos cerrados por regla de negocio	0
Número de ciclos cerrados por validaciones	0
Número de ciclos abiertos por validaciones	5
Número de ciclos abiertos por reglas de negocio	0
Número de contadores	0
Número de sumadores	0
Número de permutadotes	0
Número de cálculos por reglas de negocio	6
Número de estructuras de datos	0

Métricas Cualitativas de Construcción:

CONCEPTO	DESCRIPCIÓN
Funcionalidad	Grado de cumplimiento del objetivo para el cual fue elaborado el programa
Confiabilidad	Grado de robustez y capacidad de recuperación ante errores de captura u operación
Usabilidad	Claridad y facilidad de uso para el usuario
Eficiencia	Nivel de optimización del algoritmo
Mantenibilidad	Nivel de claridad y autodocumentación del código
Consistencia	Grado de consistencia que guarda con el diseño

Problema 2: El cuarto del Bebe

Juan y Martha se acaban de mudar a su nueva casa. Ellos tienen un pequeño bebe de tan solo 2 años de edad y quieren redecorar su habitación. Martha está preocupada porque les alcance el dinero y Juan quiere tenerlo listo antes del cumpleaños número tres del bebe que será dentro de dos semanas. Después de conversar un rato Juan decide hacer un programa de cómputo para saber cuanto cuesta y cuanto tiempo se requerirá para pintar la habitación y su techo. Dos de las paredes cuentan con dos ventanas cada una y una de ellas tiene una puerta de acceso.

Análisis

+ Identificación de Resultados

Costo total de pintar el cuarto del bebe
Tiempo que requiere pintar el cuarto del bebe

+ Identificación de Procesos

Proceso alternativo 1:

$$\begin{aligned} \text{costoTotal} &= \text{costoPintura} + \text{costoManoObra} \\ \text{tiempo} &= \text{metrosCuadrados} * \text{tiempoPorMetro} \\ \text{costoPintura} &= \text{metrosCuadrados} / \text{rendimiento} * \text{costoLitro} \\ \text{costoManoObra} &= \text{metrosCuadrados} * \text{costoMOPorMetro} \\ \text{metrosCuadrados} &= \text{metrosPared1} + \text{metrosPared2} + \text{metrosPared3} + \text{metrosPared4} + \\ &\text{metrosTecho} - \text{metrosVentana1} - \text{metrosVentana2} - \text{metrosVentana3} - \text{metrosVentana3} - \\ &\text{metrosPuerta} \\ \text{metrosPared1} &= \text{base1} * \text{altura1} \\ \text{metrosPared2} &= \text{base2} * \text{altura2} \\ \text{metrosPared3} &= \text{base3} * \text{altura3} \\ \text{metrosPared4} &= \text{base4} * \text{altura4} \\ \text{metrosTecho} &= \text{base5} * \text{altura5} \\ \text{metrosVentana1} &= \text{base6} * \text{altura6} \\ \text{metrosVentana2} &= \text{base7} * \text{altura7} \\ \text{metrosVentana3} &= \text{base8} * \text{altura8} \\ \text{metrosVentana4} &= \text{base9} * \text{altura9} \\ \text{metrosPuerta} &= \text{base10} * \text{altura10} \end{aligned}$$

Proceso alternativo 2:

$$\begin{aligned} \text{costoTotal} &= \text{costoPintura} + \text{costoManoObra} \\ \text{tiempo} &= \text{metrosCuadrados} * \text{tiempoPorMetro} \\ \text{costoPintura} &= \text{metrosCuadrados} / \text{rendimiento} * \text{costoLitro} \\ \text{costoManoObra} &= \text{metrosCuadrados} * \text{costoMOPorMetro} \\ \text{metrosCuadrados} &= \sum_{i=1}^5 \text{base}_i * \text{altura}_i - \sum_{i=6}^{10} \text{base}_i * \text{altura}_i \end{aligned}$$

+ Identificación de Datos de Entrada

Costo de Mano de Obra por metro cuadrado de pintura
Tiempo que requiere pintar un metro cuadrado
Base de la pared 1
Base de la pared 2
Base de la pared 3
Base de la pared 4
Base del techo
Base de la ventana 1 de la primera pared
Base de la ventana 2 de la primera pared
Base de la ventana 1 de la segunda pared
Base de la ventana 2 de la segunda pared
Base de la puerta
Altura de la pared 1
Altura de la pared 2
Altura de la pared 3
Altura de la pared 4
Altura del techo
Altura de la ventana 1 de la primera pared
Altura de la ventana 2 de la primera pared
Altura de la ventana 1 de la segunda pared
Altura de la ventana 2 de la segunda pared
Altura de la puerta

+ Identificación de restricciones o condiciones

El costo de la mano de obra por metro cuadrado pintado debe ser mayor a 0
El lado más largo (base) de todas las paredes, ventanas, puerta y techo debe ser mayor a 0
El lado más corto (altura) de todas las paredes, ventanas, puerta y techo debe ser mayor a 0
El área de la puerta no puede ser mayor al área de la pared en que se ubica
El área de las ventanas no puede ser mayor al área de la pared en que se ubica.

+ Repeticiones o iteraciones

Repetir la lectura de los valores de las variables de entrada en caso de que sean incorrectas.
Repetir el cálculo de las áreas de los rectángulos formados por paredes, ventanas, puerta y techo.

+ Diccionario de Datos

CostoManoObra

Nombre	Descripción	Tipo	Categoría	Dominio
costoTotal	Costo total de pintar el cuarto del bebe	Real	Salida	costoTotal > 0
costoPintura	Costo total de la pintura que se va a consumir	Real	Calculado	costoPintura > 0

Nombre	Descripción	Tipo	Categoría	Dominio
tiempo	Tiempo que requiere pintar el cuarto del bebe	Real	Salida	tiempo > 0
costoMOporMetro	Costo de Mano de Obra por metro cuadrado de pintura	Real	Entrada	costoManoObra > 0
tiempoPorMetro	Tiempo que requiere pintar un metro cuadrado	Real	Entrada	tiempoPorMetro > 0
metrosCuadrados	Total de metros cuadrados que hay que pintar	Real	Calculado	metrosCuadrados > 0
rendimiento	Cantidad de metros cuadrados que se pueden pintar con un litro de pintura	Real	Entrada	Rendimiento > 0
costoLitro	Costo de un litro de pintura	Real	Entrada	costoLitro > 0
base1*	Base de la pared 1	Real	Entrada	base1 > 0
base2*	Base de la pared 2	Real	Entrada	base2 > 0
base3*	Base de la pared 3	Real	Entrada	base3 > 0
base4*	Base de la pared 4	Real	Entrada	base4 > 0
base5*	Base del techo	Real	Entrada	base5 > 0
base6*	Base de la ventana 1 de la primera pared	Real	Entrada	base6 > 0
base7*	Base de la ventana 2 de la primera pared	Real	Entrada	base7 > 0
base8*	Base de la ventana 1 de la segunda pared	Real	Entrada	base8 > 0
base9*	Base de la ventana 2 de la segunda pared	Real	Entrada	base9 > 0
base10*	Base de la puerta	Real	Entrada	base10 > 0
altura1**	Altura de la pared 1	Real	Entrada	altura1 > 0
altura2**	Altura de la pared 2	Real	Entrada	altura2 > 0
altura3**	Altura de la pared 3	Real	Entrada	altura3 > 0
altura4**	Altura de la pared 4	Real	Entrada	altura4 > 0
altura5**	Altura del techo	Real	Entrada	altura5 > 0
altura6**	Altura de la ventana 1 de la primera pared	Real	Entrada	altura6 > 0
altura7**	Altura de la ventana 2 de la primera pared	Real	Entrada	altura7 > 0
altura8**	Altura de la ventana 1 de la segunda pared	Real	Entrada	altura8 > 0
altura9**	Altura de la ventana 2 de la segunda pared	Real	Entrada	altura9 > 0
altura10**	Altura de la puerta	Real	Entrada	altura10 > 0
metrosPared1 [†]	Metros cuadrados de la pared 1	Real	Calculado	
metrosPared2 [†]	Metros cuadrados de la pared 2	Real	Calculado	
metrosPared3 [†]	Metros cuadrados de la pared 3	Real	Calculado	
metrosPared4 [†]	Metros cuadrados de la pared 4	Real	Calculado	
metrosTecho [†]	Metros cuadrados del techo	Real	Calculado	
metrosVentana1 [†]	Metros cuadrados de la ventana 1	Real	Calculado	metrosVentana1 < metrosPared1
metrosVentana2 [†]	Metros cuadrados de la	Real	Calculado	metrosVentana2 <

Nombre	Descripción	Tipo	Categoría	Dominio
	ventana 2			metrosPared1
metrosVentana3 [†]	Metros cuadrados de la ventana 3	Real	Calculado	metrosVentana3 < metrosPared2
metrosVentana4 [†]	Metros cuadrados de la ventana 4	Real	Calculado	metrosVentana4 < metrosPared2
metrosPuerta [†]	Metros cuadrados de la puerta	Real	Calculado	metrosPuerta < metrosPared1
i	Índice que representa el número de base o altura del rectángulo en operación	entera	Calculado	1 < i < 10

* Todas estas definiciones pueden ser sustituidas por $base_i$ para el dominio de $1 < i < 5$

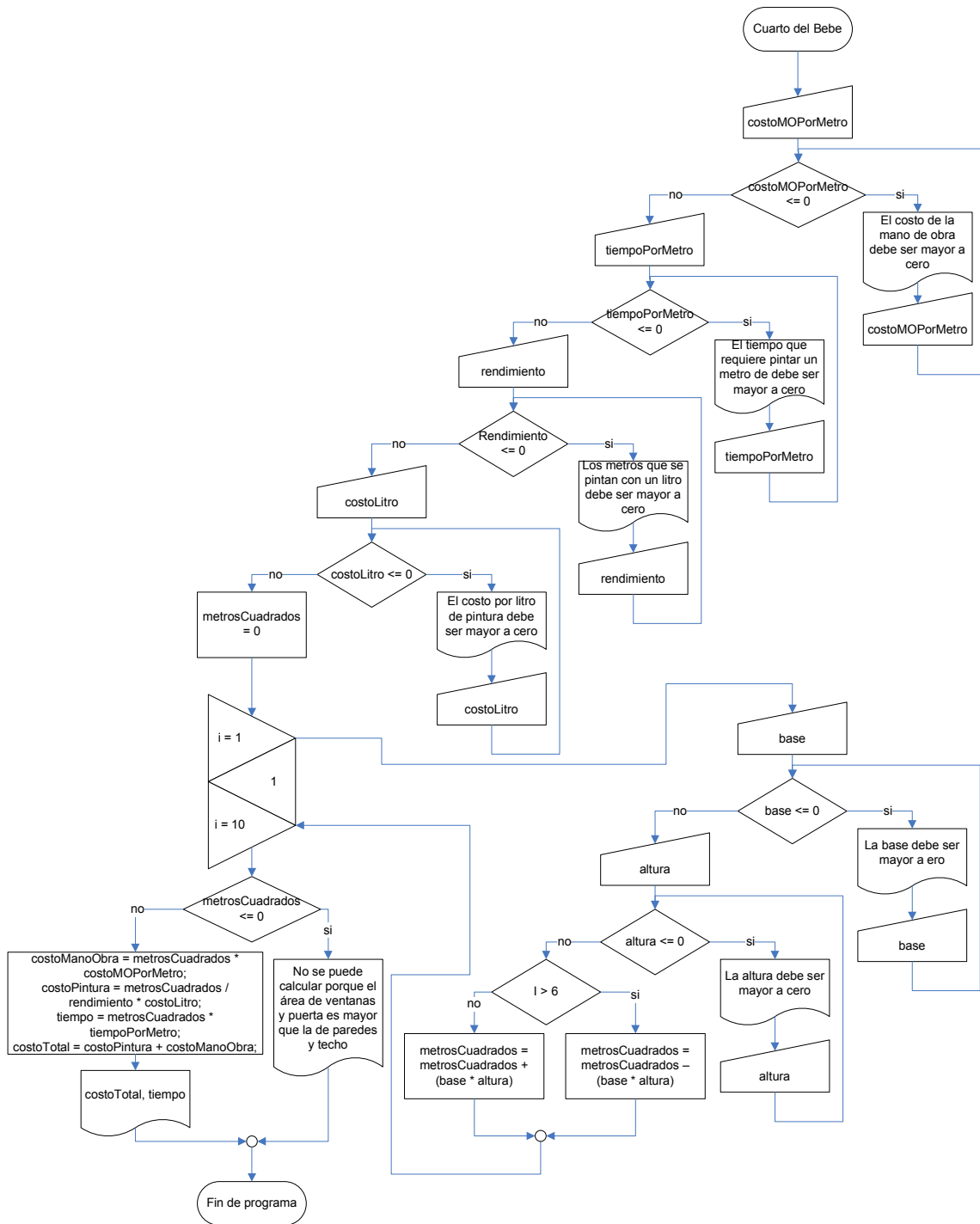
** Todas estas definiciones pueden ser sustituidas por $altura_i$ para el dominio de $6 < i < 10$

[†] Si se utiliza la formula de sumatoria, todas estas variables no son necesarias

Métricas de Análisis:

CONCEPTO	CANTIDAD	OBSERVACIONES
Resultados a obtener	2	
Procesos a realizar	5/15	Podría variar dependiendo de si se optimizo para utilizar ciclos o no.
Datos de entrada requeridos	4	
Condiciones por reglas de negocio	2	
Condiciones por validaciones	4	Estas condiciones se pueden implementar a través de ciclos abiertos
Repeticiones por validaciones	4	Corresponden a las condiciones encontradas para los datos entrada
Repeticiones por reglas de negocio	1	
Diccionario de Datos	11	

Diseño



Métricas Cuantitativas del Diseño:

CONCEPTO	CANTIDAD	OBSERVACIONES
Diseño de Secuencias	2	
Diseño de Condiciones por validaciones	6	Dependiendo del diseño la validación puede aplicarse por condición o por iteración.
Diseño de Condiciones por reglas de negocio	2	
Diseño de Iteraciones por validaciones	6	Dependiendo del diseño la validación puede aplicarse por condición o por iteración.
Diseño de iteraciones por reglas de negocio	0/1	Si el diseño es no optimizado se realizarán cada cálculo de área por separado. Si el diseño esta optimizado podría reducirse a un solo ciclo cerrado

Métricas Cualitativas del Diseño:

CONCEPTO	DESCRIPCIÓN
Pruebas	Es posible la ejecución de pruebas de caja negra y de caja blanca.
Integración	Se aplicaron correctamente las estructuras de control y las reglas de la lógica.
Consistencia	La relación entre el análisis y el diseño es consistente.

Construcción

```
using System;

namespace CuartoDelBebe
{
    /// <summary>
    /// Descripción breve de Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// Punto de entrada principal de la aplicación.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            //
            double costoTotal, costoPintura, tiempo, costoMOPorMetro,
                tiempoPorMetro, metrosCuadrados, rendimiento, costoLitro,
                lado, altura, costoManoObra;
            int i;
            Console.Write("Dime el costo de mano de obra por metro de pintura:
                ");
            costoMOPorMetro = System.Double.Parse(Console.ReadLine());
            while (costoMOPorMetro<=0)
            {
                Console.WriteLine("El costo de la MO debe ser mayor a cero");
                Console.Write("Dime el costo de mano de obra por metro de
                    pintura: ");
                costoMOPorMetro = System.Double.Parse(Console.ReadLine());
            }
        }
    }
}
```



```
Console.WriteLine("Cuanto tiempo requiere pintar un metro cuadrado? ");
tiempoPorMetro = System.Double.Parse(Console.ReadLine());
while (tiempoPorMetro<=0)
{
    Console.WriteLine("El tiempo necesario para pintar debe ser mayor
        a cero");
    Console.WriteLine("Cuanto tiempo requiere pintar un metro cuadrado?
        ");
    tiempoPorMetro = System.Double.Parse(Console.ReadLine());
}
Console.WriteLine("Cuantos metros rinde un litro de pintura? ");
rendimiento = System.Double.Parse(Console.ReadLine());
while (rendimiento<=0)
{
    Console.WriteLine("El rendimiento debe ser mayor a cero");
    Console.WriteLine("Cuantos metros rinde un litro de pintura? ");
    rendimiento = System.Double.Parse(Console.ReadLine());
}
Console.WriteLine("Cuanto cuesta el litro de pintura? ");
costoLitro = System.Double.Parse(Console.ReadLine());
while (costoLitro<=0)
{
    Console.WriteLine("El costo del litro debe ser mayor a cero");
    Console.WriteLine("Cuanto cuesta el litro de pintura? ");
    costoLitro = System.Double.Parse(Console.ReadLine());
}
metrosCuadrados=0;
for (i=1;i<=10;i++)
{
    Console.WriteLine("Dime la base del rectángulo: ");
    lado = System.Double.Parse(Console.ReadLine());
    while (lado<=0)
    {
        Console.WriteLine("La base debe ser mayor a cero");
        Console.WriteLine("Dime la base del rectángulo: ");
        lado = System.Double.Parse(Console.ReadLine());
    }
    Console.WriteLine("Dime la altura del rectángulo: ");
    altura = System.Double.Parse(Console.ReadLine());
    while (altura<=0)
    {
        Console.WriteLine("La altura debe ser mayor a cero");
        Console.WriteLine("Dime la altura del rectángulo: ");
        altura = System.Double.Parse(Console.ReadLine());
    }
    if (i>6)
    {
        metrosCuadrados = metrosCuadrados - (lado * altura);
    }
    else
    {
        metrosCuadrados = metrosCuadrados + (lado * altura);
    }
}
if (metrosCuadrados<=0)
{
    Console.WriteLine("No se puede calcular");
}
```

```

else
{
    costoManoObra = metrosCuadrados * costoMOPorMetro;
    costoPintura = metrosCuadrados / rendimiento * costoLitro;
    tiempo = metrosCuadrados * tiempoPorMetro;
    costoTotal = costoPintura + costoManoObra;
    Console.WriteLine("El tiempo requerido es de {0}", tiempo);
    Console.WriteLine("El presupuesto total es de {0}", costoTotal);
}
Console.ReadLine();
}
}
}

```

Métricas Cuantitativas de Construcción:

CONCEPTO	CANTIDAD
Número de lectura de variables validadas	6
Número de lectura de variables no validada.	0
Número de condiciones basadas en reglas del negocio	2
Número de condiciones por validaciones	0
Número de ciclos cerrados por regla de negocio	1
Número de ciclos cerrados por validaciones	0
Número de ciclos abiertos por validaciones	6
Número de ciclos abiertos por reglas de negocio	0
Número de contadores	0
Número de sumadores	0
Número de permutadotes	0
Número de cálculos por reglas de negocio	6
Número de estructuras de datos	0

Métricas Cualitativas de Construcción:

CONCEPTO	DESCRIPCIÓN
Funcionalidad	Grado de cumplimiento del objetivo para el cual fue elaborado el programa
Confiabilidad	Grado de robustez y capacidad de recuperación ante errores de captura u operación
Usabilidad	Claridad y facilidad de uso para el usuario
Eficiencia	Nivel de optimización del algoritmo
Mantenibilidad	Nivel de claridad y autodocumentación del código
Consistencia	Grado de consistencia que guarda con el diseño

Problema 3: El Viaje

Angelica es estudiante de Sistemas y quiere irse de intercambio a Canadá durante un cuatrimestre. Sabe que durante esos cuatro meses tendrá que cubrir su boleto de ida y vuelta, su colegiatura, su hospedaje y sus alimentos; además, quiere disponer de una cantidad semanal para diversiones todo esto en dólares canadienses. Durante el curso hay una semana de receso en la que quiere hacer un viaje a Nueva York para conocerlo y las agencias le proponen un paquete todo pagado en dólares americanos. Angelica quiere hacer un programa para calcular en pesos cuanto dinero necesita para su intercambio.

Análisis

+ Identificación de Resultados

Cuanto dinero necesita en total para su viaje

+ Identificación de Procesos

$$\text{presupuesto} = \text{tipoCambioDolares} * (\text{boleto} + \text{paquete}) + \text{tipoCambioCanadiense} * (4 * (\text{colegiatura} + \text{hospedaje} + \text{alimentos})) + (16 * \text{diversiones})$$

+ Identificación de Datos de Entrada

Tipo de Cambio de Dólar Americano
 Tipo de Cambio del Dólar Canadiense
 Boleto de avión redondo
 Precio del Paquete a New Cork
 Costo Mensual de Colegiatura
 Costo Mensual de Hospedaje
 Costo Mensual de alimentos
 Costo semanal para diversiones

+ Identificación de restricciones o condiciones

Todos los valores de entrada deben ser mayores a cero.

+ Repeticiones o iteraciones

Repetir la lectura de los valores de las variables de entrada en caso de que sean incorrectas.

+ Diccionario de Datos

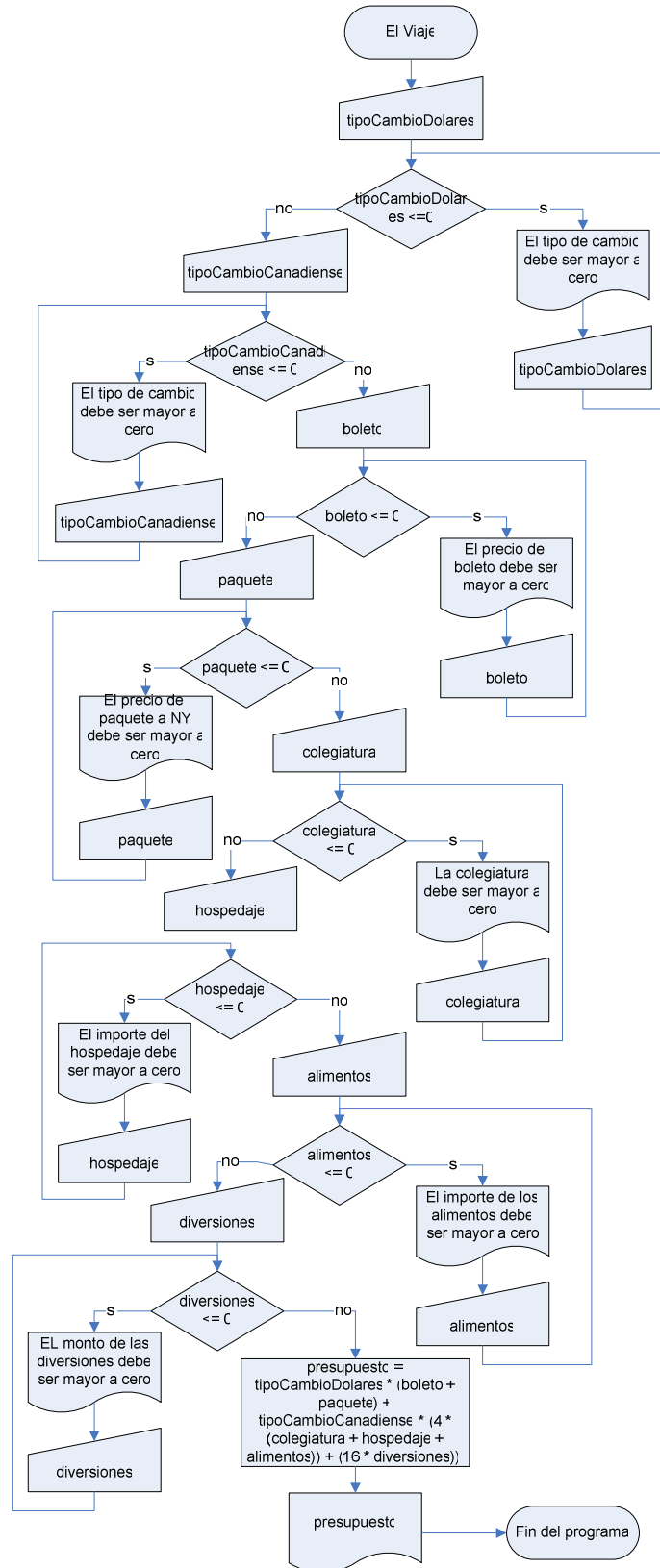
Nombre	Descripción	Tipo	Categoría	Dominio
presupuesto	Presupuesto total de sembrar pasto en el jardín	Real	Salida	Presupuesto > 0
tipoCambioDolares	Tipo de Cambio de	Real	Entrada	tipoCambioDolares > 0

Nombre	Descripción	Tipo	Categoría	Dominio
	Dólar Americano			
boleto	Boleto de avión redondo	Real	Entrada	boleto > 0
paquete	Precio del Paquete a New Cork	Real	Entrada	paquete > 0
tipoCambioCanadiense	Tipo de Cambio del Dólar Canadiense	Real	Entrada	tipoCambioCanadiense > 0
colegiatura	Costo Mensual de Colegiatura	Real	Entrada	colegiatura > 0
hospedaje	Costo Mensual de Hospedaje	Real	Entrada	hospedaje > 0
alimentos	Costo Mensual de alimentos	Real	Entrada	alimentos > 0
diversiones	Costo semanal para diversiones	Real	Entrada	Diversiones > 0

Métricas de Análisis:

CONCEPTO	CANTIDAD	OBSERVACIONES
Resultados a obtener	1	
Procesos a realizar	1	Podría variar el número dependiendo de la fórmula diseñada para el cálculo. El mínimo serían uno pero pueden ser mas.
Datos de entrada requeridos	8	
Condiciones por reglas de negocio	0	
Condiciones por validaciones	8	Estas condiciones se pueden implementar a través de ciclos abiertos
Repeticiones por validaciones	8	Estas condiciones se pueden implementar a través de condiciones
Repeticiones por reglas de negocio	0	
Diccionario de Datos	9	

Diseño



Métricas Cuantitativas del Diseño:

CONCEPTO	CANTIDAD	OBSERVACIONES
Diseño de Secuencias	1	
Diseño de Condiciones por validaciones	8	Dependiendo del diseño la validación puede aplicarse por condición o por iteración.
Diseño de Condiciones por reglas de negocio	0	
Diseño de Iteraciones por validaciones	8	Dependiendo del diseño la validación puede aplicarse por condición o por iteración.
Diseño de iteraciones por reglas de negocio	0	

Métricas Cualitativas del Diseño:

CONCEPTO	DESCRIPCIÓN
Pruebas	Es posible la ejecución de pruebas de caja negra y de caja blanca.
Integración	Se aplicaron correctamente las estructuras de control y las reglas de la lógica.
Consistencia	La relación entre el análisis y el diseño es consistente.

Construcción:

```
using System;

namespace Viaje
{
    /// <summary>
    /// Descripción breve de Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// Punto de entrada principal de la aplicación.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            //
            double presupuesto, tipoCambioDolares, tipoCambioCanadiense,
                boleto,
                paquete, colegiatura, hospedaje, alimentos, diversiones;
            int i;
            Console.Write("Dime a como está el tipo de cambio del dólar
                americano: ");
            tipoCambioDolares = System.Double.Parse(Console.ReadLine());
            while (tipoCambioDolares<=0)
            {
                Console.WriteLine("El tipo de cambio debe ser mayor a cero");
                Console.Write("Dime a como está el tipo de cambio del dólar
                    americano: ");
                tipoCambioDolares = System.Double.Parse(Console.ReadLine());
            }
            Console.Write("Dime a como está el tipo de cambio del dólar
                canadiense: ");
            tipoCambioCanadiense = System.Double.Parse(Console.ReadLine());
            while (tipoCambioCanadiense<=0)
```

```
{
    Console.WriteLine("El tipo de cambio debe ser mayor a cero");
    Console.Write("Dime a como está el tipo de cambio del dólar
        canadiense: ");
    tipoCambioCanadiense = System.Double.Parse(Console.ReadLine());
}
Console.Write("Cuanto cuesta el boleto redondo? ");
boleto = System.Double.Parse(Console.ReadLine());
while (boleto<=0)
{
    Console.WriteLine("El costo del boleto debe ser mayor a cero");
    Console.Write("Cuanto cuesta el boleto redondo? ");
    boleto = System.Double.Parse(Console.ReadLine());
}
Console.Write("Cuanto cuesta el paquete a New York? ");
paquete = System.Double.Parse(Console.ReadLine());
while (paquete<=0)
{
    Console.WriteLine("El costo del paquete debe ser mayor a cero");
    Console.Write("Cuanto cuesta el paquete a New York? ");
    paquete = System.Double.Parse(Console.ReadLine());
}
Console.Write("Cuanto cuesta la colegiatura mensual? ");
colegiatura = System.Double.Parse(Console.ReadLine());
while (colegiatura<=0)
{
    Console.WriteLine("El costo de la colegiatura debe ser mayor a
        cero");
    Console.Write("Cuanto cuesta la colegiatura mensual? ");
    colegiatura = System.Double.Parse(Console.ReadLine());
}
Console.Write("Cuanto cuesta el hospedaje mensual? ");
hospedaje = System.Double.Parse(Console.ReadLine());
while (hospedaje<=0)
{
    Console.WriteLine("El costo del hospedaje ser mayor a cero");
    Console.Write("Cuanto cuesta el hospedaje mensual? ");
    hospedaje = System.Double.Parse(Console.ReadLine());
}
Console.Write("Cuanto cuesta la alimentación mensual? ");
alimentos = System.Double.Parse(Console.ReadLine());
while (alimentos<=0)
{
    Console.WriteLine("El costo de los alimentos debe ser mayor a
        cero");
    Console.Write("Cuanto cuesta la alimentación mensual? ");
    alimentos = System.Double.Parse(Console.ReadLine());
}
Console.Write("Cuanto quieres gastar de diversiones a la
    semana? ");
diversiones = System.Double.Parse(Console.ReadLine());
while (diversiones<=0)
{
    Console.WriteLine("El costo de las diversiones debe ser mayor a
        cero");
    Console.Write("Cuanto quieres gastar de diversiones a la
        semana? ");
    diversiones = System.Double.Parse(Console.ReadLine());
}
```

```

    }
    presupuesto=tipoCambioDolares * (boleto+paquete) +
        tipoCambioCanadiense *(4*(colegiatura + hospedaje +
            alimentos)) + (16*diversiones);
    Console.WriteLine("EL presupuesto total para le viaje es
        {0}",presupuesto);
    Console.ReadLine();

    //
}
}
}

```

Métricas Cuantitativas de Construcción:

CONCEPTO	CANTIDAD
Número de lectura de variables validadas	8
Número de lectura de variables no validada.	0
Número de condiciones basadas en reglas del negocio	0
Número de condiciones por validaciones	8
Número de ciclos cerrados por regla de negocio	0
Número de ciclos cerrados por validaciones	0
Número de ciclos abiertos por validaciones	8
Número de ciclos abiertos por reglas de negocio	0
Número de contadores	0
Número de sumadores	0
Número de permutadotes	0
Número de cálculos por reglas de negocio	1
Número de estructuras de datos	0

Métricas Cualitativas de Construcción:

CONCEPTO	DESCRIPCIÓN
Funcionalidad	Grado de cumplimiento del objetivo para el cual fue elaborado el programa
Confiabilidad	Grado de robustez y capacidad de recuperación ante errores de captura u operación
Usabilidad	Claridad y facilidad de uso para el usuario
Eficiencia	Nivel de optimización del algoritmo
Mantenibilidad	Nivel de claridad y autodocumentación del código
Consistencia	Grado de consistencia que guarda con el diseño

Problema 4: La Escuela

Andrés está haciendo su servicio social en una pequeña escuela primaria como asistente de la coordinación académica y le han pedido que obtenga una estadística para saber cuál es el promedio global de calificaciones de los 25 alumnos de 6° año de primaria así como cuántos alumnos están reprobados, cuántos tienen un promedio entre 6 y 7.9 y cuántos tienen promedio arriba de 8. Cada estudiante lleva doce materias. Andrés pensó que es mejor hacer un programa que lo ayude a obtener esa estadística que hacer los cálculos a mano.

Análisis

+ Identificación de Resultados

Promedio global de calificaciones de 25 estudiantes

Cuántos alumnos están reprobados

Cuántos tienen un promedio entre 6 y 7.9

Cuántos tienen promedio arriba de 8

+ Identificación de Procesos

Proceso alternativo 1:

$$\text{promedioGlobal} = (\text{promedio1} + \text{promedio2} + \text{promedio3} + \text{promedio4} + \text{promedio5} + \text{promedio6} + \text{promedio7} + \text{promedio8} + \text{promedio9} + \text{promedio10} + \text{promedio11} + \text{promedio12} + \text{promedio13} + \text{promedio14} + \text{promedio15} + \text{promedio16} + \text{promedio17} + \text{promedio18} + \text{promedio19} + \text{promedio20} + \text{promedio21} + \text{promedio22} + \text{promedio23} + \text{promedio23} + \text{promedio25}) / 25$$

$$\text{promedio1} = (\text{calificacion1} + \text{calificacion2} + \text{calificacion3} + \text{calificacion4} + \text{calificacion5} + \text{calificacion6} + \text{calificacion7} + \text{calificacion8} + \text{calificacion9} + \text{calificacion10} + \text{calificacion11} + \text{calificacion12}) / 12$$

Si promedio1 es menor que seis, reprobados = reprobados + 1

Si promedio1 esta entre 6 y 7.9, aprobadoMedio = aprobadoMedio + 1

Si promedio1 es ocho o mayor, aprobadoAlto = aprobadoAlto + 1

$$\text{promedio2} = (\text{calificacion1} + \text{calificacion2} + \text{calificacion3} + \text{calificacion4} + \text{calificacion5} + \text{calificacion6} + \text{calificacion7} + \text{calificacion8} + \text{calificacion9} + \text{calificacion10} + \text{calificacion11} + \text{calificacion12}) / 12$$

Si promedio2 es menor que seis, reprobados = reprobados + 1

Si promedio2 esta entre 6 y 7.9, aprobadoMedio = aprobadoMedio + 1

Si promedio2 es ocho o mayor, aprobadoAlto = aprobadoAlto + 1

$$\text{promedio3} = (\text{calificacion1} + \text{calificacion2} + \text{calificacion3} + \text{calificacion4} + \text{calificacion5} + \text{calificacion6} + \text{calificacion7} + \text{calificacion8} + \text{calificacion9} + \text{calificacion10} + \text{calificacion11} + \text{calificacion12}) / 12$$

Si promedio3 es menor que seis, reprobados = reprobados + 1

Si promedio3 esta entre 6 y 7.9, aprobadoMedio = aprobadoMedio + 1

Si promedio3 es ocho o mayor, aprobadoAlto = aprobadoAlto + 1

...

$promedio25 = (calificacion1 + calificacion2 + calificacion3 + calificacion4 + calificacion5 + calificacion6 + calificacion7 + calificacion8 + calificacion9 + calificacion10 + calificacion11 + calificacion12) / 12$

Si promedio25 es menor que seis, reprobados = reprobados + 1

Si promedio25 esta entre 6 y 7.9, aprobadoMedio = aprobadoMedio + 1

Si promedio25 es ocho o mayor, aprobadoAlto = aprobadoAlto + 1

Proceso alternativo 2:

Repetir i desde 1 hasta 25 (para cada estudiante)

$$promedioGlobal = \frac{\sum_{i=1}^{25} promedio_i}{25}$$

$$promedio_i = \frac{\sum_{j=1}^{12} calificacion_j}{12}$$

reprobados = reprobados + 1, Si promedio_i es menor que seis

aprobadoMedio = aprobadoMedio + 1, Si promedio_i esta entre 6 y 7.9

aprobadoAlto = aprobadoAlto + 1, Si promedio_i es ocho o mayor

+ Identificación de Datos de Entrada

Calificación 1 del primer estudiante

Calificación 2 del primer estudiante

Calificación 3 del primer estudiante

Calificación 4 del primer estudiante

Calificación 5 del primer estudiante

Calificación 6 del primer estudiante

Calificación 7 del primer estudiante

Calificación 8 del primer estudiante

Calificación 9 del primer estudiante

Calificación 10 del primer estudiante

Calificación 11 del primer estudiante

Calificación 12 del primer estudiante

Calificación 1 del primer estudiante

Calificación 2 del segundo estudiante

Calificación 3 del segundo estudiante

Calificación 4 del segundo estudiante

Calificación 5 del segundo estudiante

Calificación 6 del segundo estudiante

Calificación 7 del segundo estudiante

Calificación 8 del segundo estudiante

Calificación 9 del segundo estudiante

Calificación 10 del segundo estudiante

Calificación 11 del segundo estudiante

Calificación 12 del segundo estudiante

...

Calificación 1 del último estudiante

Calificación 2 del último estudiante

Calificación 3 del último estudiante

Calificación 4 del último estudiante

Calificación 5 del último estudiante

Calificación 6 del último estudiante

Calificación 7 del último estudiante

Calificación 8 del último estudiante

Calificación 9 del último estudiante

Calificación 10 del último estudiante

Calificación 11 del último estudiante

Calificación 12 del último estudiante

* se puede escribir Calificación_i del estudiante_i

+ Identificación de restricciones o condiciones

Todas las calificaciones debes estar entre 0 y 10

Si promedio_i es menor que seis, reprobados = reprobados + 1

Si promedio_i esta entre 6 y 7.9, aprobadoMedio = aprobadoMedio + 1

Si promedio_i es ocho o mayor, aprobadoAlto = aprobadoAlto + 1

+ Repeticiones o iteraciones

Repetir la lectura de 12 calificaciones para cada uno de los 25 estudiantes

+ Diccionario de Datos

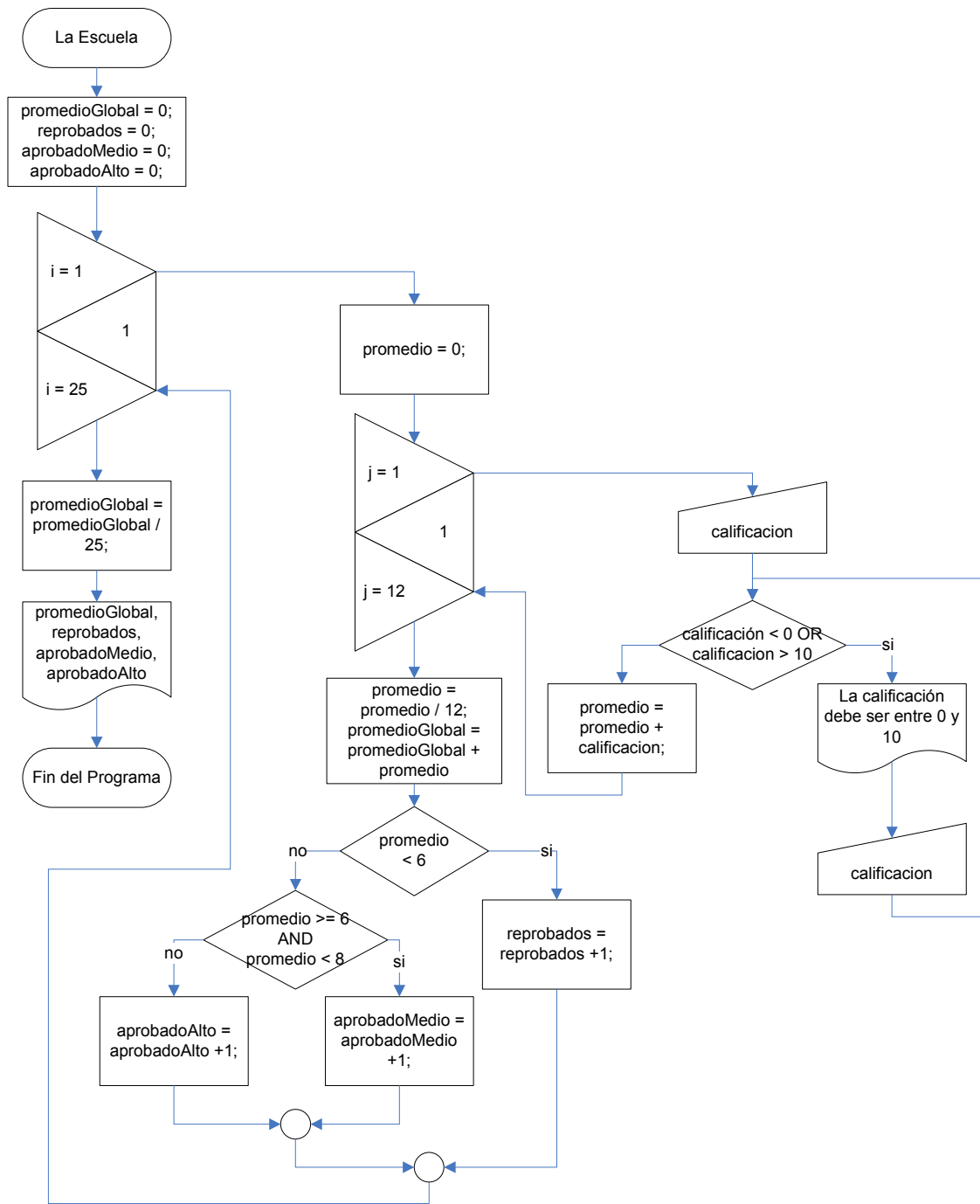
Nombre	Descripción	Tipo	Categoría	Dominio
promedioGlobal	Promedio global de calificaciones de 25 estudiantes	Real	Salida	0 < promedioGlobal <= 10
promedio	Promedio de calificaciones de un solo alumno	Real	Calculado	0 < promedio <= 10
reprobados	Cuantos alumnos están reprobados	Integer	Salida	
aprobadoMedio	Cuantos tienen un promedio entre 6 y 7.9	Integer	Salida	
aprobadoAlto	Cuantos tienen promedio arriba de 8	Integer	Salida	
calificacion _i *	Tiempo que requiere pintar un metro cuadrado	Real	Entrada	0 < calificacion _i <= 10

* equivale a la sucesión de calificación1, calificación2, ..., calificación12 para cada estudiante.

Métricas de Análisis:

CONCEPTO	CANTIDAD	OBSERVACIONES
Resultados a obtener	4	
Procesos a realizar	7/n	Este proceso debe estar optimizado, si no es así podrían ser mas de 300
Datos de entrada requeridos	1/300	Se puede optimizar y realizar solo una lectura, de lo contrario se requerirían 300 lecturas
Condiciones por reglas de negocio	3	Aplicado a los contadores
Condiciones por validaciones	1	Estas condiciones se pueden implementar a través de ciclos abiertos
Repeticiones por validaciones	1	Corresponden a las condiciones encontradas para los datos entrada
Repeticiones por reglas de negocio	2	
Diccionario de Datos	6	Uno de ellos, las calificaciones son 300 instancias del mismo tipo de dato.

Diseño



Métricas Cuantitativas del Diseño:

CONCEPTO	CANTIDAD	OBSERVACIONES
Diseño de Secuencias	5	Cuatro de contadores una de calificaciones
Diseño de Condiciones por validaciones	1	Dependiendo del diseño la validación puede aplicarse por condición o por iteración.
Diseño de Condiciones por reglas de negocio	2	
Diseño de Iteraciones por validaciones	1	Dependiendo del diseño la validación puede aplicarse por condición o por iteración.
Diseño de iteraciones por reglas de negocio	2	

Métricas Cualitativas del Diseño:

CONCEPTO	DESCRIPCIÓN
Pruebas	Es posible la ejecución de pruebas de caja negra y de caja blanca.
Integración	Se aplicaron correctamente las estructuras de control y las reglas de la lógica.
Consistencia	La relación entre el análisis y el diseño es consistente.

Construcción

```
using System;

namespace Escuela
{
    /// <summary>
    /// Descripción breve de Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// Punto de entrada principal de la aplicación.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            //
            double promedioGlobal, calificacion, promedio;
            int i,j,reprobados, aprobadoMedio, aprobadoAlto;
            promedioGlobal=0;
            reprobados=0;
            aprobadoMedio=0;
            aprobadoAlto=0;
            for (i=1;i<=25;i++)
            {
                promedio=0;
                for (j=1;j<=12;j++)
                {
                    Console.WriteLine("Dime la {0}a Calificación del alumno No. {1}:
                    ", j, i);
                    calificacion = System.Double.Parse(Console.ReadLine());
                    while (calificacion<0 || calificacion>10)
```

```

    {
        Console.WriteLine("La calificación debe ser entre cero y
            diez");
        Console.Write("Dime la {0}a Calificación del alumno No. {1}:
            ", j, i);
        calificacion = System.Double.Parse(Console.ReadLine());
    }
    promedio=promedio+calificacion;
}
promedio = promedio /12;
promedioGlobal = promedioGlobal+promedio;
if (promedio<6)
{
    reprobados++;
}
else
{
    if (promedio>=6 && promedio<8)
    {
        aprobadoMedio++;
    }
    else
    {
        aprobadoAlto++;
    }
}
}
promedioGlobal=promedioGlobal/25;
Console.WriteLine("El promedio Global es {0}",promedioGlobal);
Console.WriteLine("El número de reprobados es {0}",reprobados);
Console.WriteLine("El número de alumnos que sacaron entre 6 y 8 es
    {0}", aprobadoMedio);
Console.WriteLine("El número de alumnos que sacaron más de 8 es
    {0}", aprobadoAlto);
Console.ReadLine();
//
}
}
}

```

Métricas Cuantitativas de Construcción:

CONCEPTO	CANTIDAD
Número de lectura de variables validadas	1
Número de lectura de variables no validada.	0
Número de condiciones basadas en reglas del negocio	2
Número de condiciones por validaciones	1
Número de ciclos cerrados por regla de negocio	2
Número de ciclos cerrados por validaciones	0
Número de ciclos abiertos por validaciones	1
Número de ciclos abiertos por reglas de negocio	0
Número de contadores	3
Número de sumadores	2
Número de permutadotes	0
Número de cálculos por reglas de negocio	2
Número de estructuras de datos	0

Métricas Cualitativas de Construcción:

CONCEPTO	DESCRIPCIÓN
Funcionalidad	Grado de cumplimiento del objetivo para el cual fue elaborado el programa
Confiabilidad	Grado de robustez y capacidad de recuperación ante errores de captura u operación
Usabilidad	Claridad y facilidad de uso para el usuario
Eficiencia	Nivel de optimización del algoritmo
Mantenibilidad	Nivel de claridad y autodocumentación del código
Consistencia	Grado de consistencia que guarda con el diseño

CONCENTRADO

Métricas de Análisis:

CONCEPTO	EL JARDÍN	CUARTO DEL BEBE	EL VIAJE	LA ESCUELA
Resultados a obtener	2	2	1	4
Procesos a realizar	6	5/15	1	7/11
Datos de entrada requeridos	5	4	8	1/300
Condiciones por reglas de negocio	1	2	0	3
Condiciones por validaciones	5	4	8	1
Repeticiones por validaciones	5	4	8	1
Repeticiones por reglas de negocio	0	1	0	2
Diccionario de Datos	11	11	9	6

Métricas Cuantitativas del Diseño:

CONCEPTO	EL JARDÍN	CUARTO DEL BEBE	EL VIAJE	LA ESCUELA
Diseño de Secuencias	2	2	1	5
Diseño de Condiciones por validaciones	5	6	8	1
Diseño de Condiciones por reglas de negocio	1	2	0	2
Diseño de Iteraciones por validaciones	5	6	8	1
Diseño de iteraciones por reglas de negocio	0	0/1	0	2

Métricas Cualitativas del Diseño:

CONCEPTO	DESCRIPCIÓN
Pruebas	Es posible la ejecución de pruebas de caja negra y de caja blanca.
Integración	Se aplicaron correctamente las estructuras de control y las reglas de la lógica.
Consistencia	La relación entre el análisis y el diseño es consistente.

Métricas Cuantitativas de Construcción:

CONCEPTO	EL JARDÍN	CUARTO DEL BEBE	EL VIAJE	LA ESCUELA
Número de lectura de variables validadas	5	6	8	1
Número de lectura de variables no validada.	0	0	0	0
Número de condiciones basadas en reglas del negocio	1	2	0	2
Número de condiciones por validaciones	0	0	8	1
Número de ciclos cerrados por regla de negocio	0	1	0	2
Número de ciclos cerrados por validaciones	0	0	0	0
Número de ciclos abiertos por validaciones	5	6	8	1
Número de ciclos abiertos por reglas de negocio	0	0	0	0
Número de contadores	0	0	0	3
Número de sumadores	0	0	0	2
Número de permutadotes	0	0	0	0
Número de cálculos por reglas de negocio	6	6	1	2
Número de estructuras de datos	0	0	0	0

Métricas Cualitativas de Construcción:

CONCEPTO	DESCRIPCIÓN
Funcionalidad	Grado de cumplimiento del objetivo para el cual fue elaborado el programa
Confiabilidad	Grado de robustez y capacidad de recuperación ante errores de captura u operación
Usabilidad	Claridad y facilidad de uso para el usuario
Eficiencia	Nivel de optimización del algoritmo
Mantenibilidad	Nivel de claridad y autodocumentación del código
Consistencia	Grado de consistencia que guarda con el diseño

Anexo C-4

Formato para concentrar datos de la aplicación

Folio:

Total:

ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS		DISEÑO		CODIFICA.		

Folio:

Total:

ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS		DISEÑO		CODIFICA.		

Folio:

Total:

ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS		DISEÑO		CODIFICA.		

Folio:

Total:

ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS		DISEÑO		CODIFICA.		

Folio:

Total:

ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS		DISEÑO		CODIFICA.		

Folio:

Total:

ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS	Resultados	Procesos	Entradas	Restricciones	Repeticiones	Diccionario
DISEÑO	Secuencias	Condiciones	Repeticiones	Pruebas	Integración	Consistencia
CODIFICACIÓN	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenible	Consistencia
ANÁLISIS		DISEÑO		CODIFICA.		