UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

Centro de Ciencias Básicas

Teoría de Diseño de Sistemas de Información para Sistemas Visualizadores de Protocolos Verbales para el Aprendizaje de la Programación Básica

TESIS

Que para obtener el grado de
Doctor en Ciencias Exactas, Sistemas y de la Información

PRESENTA

M.I.T.C Carlos Argelio Arévalo Mercado

DIRECTOR DE TESIS

Juan Manuel Gómez Reynoso, Ph.D.

ASESORES

Dr. Francisco Javier Álvarez Rodríguez

Dr. José Manuel Mora Tavarez

Aguascalientes, Ags. Junio, 2010

Agradecimientos

A Dios, cuyos designios misteriosos no logro entender, pero a quien agradezco todos los días permitirme experimentar la maravilla de la vida y la conciencia.

A la Universidad Autónoma de Aguascalientes, noble Institución donde trabajo, que me proporcionó el financiamiento y las facilidades para cursar estos estudios de Doctorado.

A mi esposa Claudia, por su paciencia y total entrega en estos difíciles años. Sin su fortaleza, nuestra familia no se hubiera podido mantener unida. Te amo corazón.

A mi Madre, sin cuyo soporte amoroso, sencillamente nosotros, la familia Arévalo García, no hubiésemos podido salir adelante. El cariño de una madre no conoce límites.

A mi asesor, el Dr. Juan Manuel Gómez Reynoso, por su dedicada enseñanza de los métodos y formas del rigor científico que caracterizan a una investigación de esta naturaleza.

A la Dra. Lizbeth Muñoz Andrade, por su ayuda sincera, leal y constante durante esta época de estudio.

A los profesores Jesús Palos, Lorena Pinales y Blanca Estrada (y a sus alumnos), por auxiliarme con las pruebas experimentales descritas en esta tesis.

Al L.I. Jesús Héctor Ruiz Gallegos, por ayudarme pacientemente con el diseño y construcción de los prototipos animados.

Al I.S.C. José Ramón Díaz Navarrete, por su crítica ayuda en la programación del prototipo visor de protocolos basado en Web.

A la Maestra Lizeth Iztiguery Solano Romo por su amable ayuda con la prueba piloto de usabilidad y por sus ánimos.

A la L.I. Anaid Macias, por su ayuda en la investigación y transcripción de material instruccional sobre programación.

A mis sinodales, por proporcionarme valiosa retroalimentación para mejorar la calidad del estudio.

A los alumnos de 8º semestre de L.T.I. de la UAA (Enero-Junio 2010) por ayudarme con la prueba de usabilidad.

Al cuerpo Académico GTIS coordinado por el Dr. Manuel Mora Tavarez, por brindar el apoyo económico para las estancias y visitas a congresos efectuados durante esta investigación.

A mis compadres y entrañables amigos, el Lic. Enrique Rodríguez Hernández y Naty López Soto, por mostrarnos el significado de la amistad incondicional.

Dedicatorias

A mis hijos, para que este esfuerzo los inspire a alcanzar sus propias metas, que seguramente opacarán con creces este sencillo logro.

A Claudia Andrea, que estas a punto de iniciar sus estudios universitarios. Que en esa nueva fase de tu vida encuentres la felicidad que da el identificar una vocación, para servicio de tus semejantes.

A Carla Argelia, que pronto iniciarás tus estudios de secundaria. Que en esta época aumentes la tremenda capacidad que ya tienes y no pierdas nunca la belleza y ternura de tu carácter.

A Carlos Arturo, el Ángel que Dios nos ha mandado para mostrarnos todos los días lo hermosa que es la inocencia del alma infantil. Que juntos como familia podamos continuar acompañándote en tu camino.

A mis suegros, Don Arturo y Lupita, por abrirnos todos los días las puertas de su casa y arropar a nuestros hijos con paciencia y cariño.

A mi hermano, el Dr. Alejandro Arévalo Mercado. La vida ha hecho que últimamente nos veamos poco, pero por nada del mundo pasaré por alto incluirte cariñosamente en esta dedicatoria. ¡Tal vez ahora tengamos más tiempo!

A mi cuñado, Adrian Alejandro, esperando desarrolles plenamente tu potencial profesional y personal, para alegría de todos los que te queremos.

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

Centro de Ciencias Básicas

M. I.T.C. CARLOS ARGELIO ARÉVALO MERCADO
PASANTE DEL DOCTORADO EN CIENCIAS EXACTAS, SISTEMAS Y DE LA INFORMACIÓN
P R E S E N T E .

Estimado (a) Alumno (a) Arévalo:

Por medio de este conducto me permito comunicar a Usted que habiendo recibido los votos aprobatorios de los revisores de su trabajo de tesis titulado: **"Teoría de Diseño de Sistemas de Información para Sistemas Visualizadores de Protocolos Verbales para el aprendizaje de la programación básica"**, hago de su conocimiento que puede imprimir dicho documento y continuar con los trámites para la presentación de su examen de grado.

Sin otro particular me permito saludarle muy afectuosamente.

A T E N T A M E N T E
Aguascalientes, Ags., 11 de junio de 2010
"LUMEN PROFERRE"
EL DECANO

DR. FRANCISCO JAVIER ÁLVAREZ RODRÍGUEZ

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES
CENTRO DE CIENCIAS BASICAS

c.c.p.- Archivo

Dr. **Francisco Javier Álvarez Rodríguez**
Decano del Centro de Ciencias Básicas

Por medio de este conducto, autorizo al tesista:

M.I.T.C. CARLOS ARGELIO AREVALO MERCADO

la impresión de su documento de tesis con titulo **"Teoría de Diseño de Sistemas de Información para Sistemas Visualizadores de Protocolos Verbales para el aprendizaje de la programación básica"**, ya que cumple con los requisitos de contenido y forma exigidas en la Universidad Autónoma de Aguascalientes.

Sin más por el momento, quedo a sus órdenes para cualquier aclaración al respecto.

A T E N T A M E N T E

Dr. Juan Manuel Gómez Reynoso
Asesor

Aguascalientes, Ags., a 8 de Junio del 2010

DR. FRANCISCO JAVIER ÁLVAREZ RODRÍGUEZ
DECANO DEL CENTRO DE CIENCIAS BÁSICAS
P R E S E N T E.

Por medio de este conducto, autorizo al tesista:

M.I.T.C. CARLOS ARGELIO ARÉVALO MERCADO

la impresión de su documento de tesis con título "Teoría de Diseño de Sistemas de Información para Sistemas Visualizadores de Protocolos Verbales para el aprendizaje de la programación básica", ya que cumple con los requisitos de contenido y forma exigidas en la Universidad Autónoma de Aguascalientes.

Sin más por el momento, me permito saludarle muy afectuosamente.

A T E N T A M E N T E
Aguascalientes, Ags., 11 de junio de 2010
"SE LUMEN PROFERRE"

DR. FRANCISCO JAVIER ÁLVAREZ RODRÍGUEZ
ASESOR DE TESIS

Dr. Francisco Javier Álvarez Rodriguez
Decano del Centro de Ciencias Básicas

Por medio de este conducto, autorizo al tesista:

M.I.T.C. CARLOS ARGELIO AREVALO MERCADO

la impresión de su documento de tesis con título **"Teoria de Diseño de Sistemas de Información para Sistemas Visualizadores de Protocolos Verbales para el aprendizaje de la programación básica"**, ya que cumple con los requisitos de contenido y forma exigidas en la Universidad Autónoma de Aguascalientes.

Sin más por el momento, quedo a sus órdenes para cualquier aclaración al respecto.

A T E N T A M E N T E

Dr. José Manuel Mora Tavares
Asesor

Aguascalientes, Ags., a 8 de Junio del 2010

Dr. Francisco Javier Álvarez Rodríguez
Decano del Centro de Ciencias Básicas

Por medio de este conducto, autorizo al tesista:

M.I.T.C. CARLOS ARGELIO AREVALO MERCADO

la impresión de su documento de tesis con título **"Teoría de Diseño de Sistemas de Información para Sistemas Visualizadores de Protocolos Verbales para el aprendizaje de la programación básica"**, ya que cumple con los requisitos de contenido y forma exigidas en la Universidad Autónoma de Aguascalientes.

Sin más por el momento, quedo a sus órdenes para cualquier aclaración al respecto.

ATENTAMENTE

Dra. Laura A. Garza González
Asesor

Aguascalientes, Ags., a 8 de Junio del 2010

Dr. Francisco Javier Álvarez Rodríguez
Decano del Centro de Ciencias Básicas

Por medio de este conducto, autorizo al tesista:

M.I.T.C. CARLOS ARGELIO AREVALO MERCADO

la impresión de su documento de tesis con título "**Teoría de Diseño de Sistemas de Información para Sistemas Visualizadores de Protocolos Verbales para el aprendizaje de la programación básica**", ya que cumple con los requisitos de contenido y forma exigidas en la Universidad Autónoma de Aguascalientes.

Sin más por el momento, quedo a sus órdenes para cualquier aclaración al respecto.

A T E N T A M E N T E

Dra. Alma Elena Figueroa Rubalcava
Asesor

Aguascalientes, Ags., a 8 de Junio del 2010

Resumen

Aprender a programar es difícil para muchos estudiantes universitarios de primer año, tanto a nivel local como internacional (Bayman 1983; T. Boyle 2003; Dijkstra 1989; Jenkins 2002; UAA 2007). La literatura señala varios factores que pueden influenciar el éxito en el primer curso de programación: la experiencia previa, los modelos mentales, la habilidad para las matemáticas y la autoeficacia, entre otros. Sin embargo aún no se han identificado claros predictores de éxito (Bergin 2005; P. Byrne 2001; Simon et al. 2006; Chenglie Hu 2006).

Las estrategias instruccionales en los cursos tradicionales de programación tienden a enfocarse hacia aspectos de sintaxis. (Linn y Clancy 1992; McIver 2001; A. Robins, J. Rountree, y N. Rountree 2003).

La literatura cognitiva que estudia los procesos mentales involucrados en el proceso de programar, sugiere que los modelos mentales válidos, las estrategias de solución de problemas (llamadas planes o esquemas) y la metacognición, son elementos presentes en programadores expertos y que influyen significativamente en su habilidad para programar (Brooks 1990; Fixx, Wiedenbeck, y Scholtz 1993; Machanick 2005; Rist 1995; Soloway y Ehrlich 1984). En clases tradicionales, esta habilidad estratégica se enseña normalmente explicando paso por paso el proceso de escribir un programa (A. Robins, J. Rountree, y N. Rountree 2003).

La mayoría de las aplicaciones de software orientadas a apoyar el proceso de aprendizaje de la programación, se enfocan en la creación de modelos mentales válidos, basándose en animaciones, interactividad y tecnología multimedia (Guzdial 2002; Hamer 2004; McKeown 2004; Moskal, Lurie, y Stephen Cooper 2004; Pollack 2004). Se ha encontrado que existen pocas aplicaciones diseñadas para mostrar explícitamente a un programador novato el conocimiento estratégico y metacognitivo de un experto, al momento de escribir un programa.

Los protocolos verbales (Ericsson y Simon 1993) se usan en la Ingeniería de software como técnica para registrar los procesos cognitivos que tienen lugar en la memoria de corto plazo de un usuario o de un experto al momento de evaluar una aplicación o de resolver un problema. En la presente tesis se argumenta que un protocolo verbal puede ser usado como un mecanismo para mostrar el conocimiento estratégico y metacognitivo de un instructor al escribir un programa básico. La teoría de la carga cognitiva (J. Sweller 1988a; John Sweller 1994) sugiere estrategias instruccionales (llamadas efectos) que toman en cuenta las limitaciones intrínsecas del cerebro humano. En el contexto de tal teoría, se ha documentado que el uso de ejemplos resueltos y de problemas por completar son estrategias instruccionales efectivas para enseñar a estudiantes principiantes. (Gerjets, Scheiter, y Catrambone 2004; Renkl, Atkinson, y Große 2004; Renkl, Hilbert, y Schworm 2009). La teoría de Codificación Dual (Paivio 1990; Sadoski y Paivio 2004) señala que la mente humana es capaz de retener y recuperar información y conocimiento si éstos se presentan en una modalidad dual, por medio de lenguaje verbal e imágenes.

En la presente tesis se aplicó el marco metodológico conocido como Teoría de Diseño de Sistemas de Información (Information Systems Design Theory o ISDT por sus siglas en inglés) propuesto por Walls, et. al (1992). Bajo este marco de referencia, se propuso una teoría de diseño para un tipo especial de artefacto de software nombrado como Sistema Visor de Protocolos, que busca fomentar la transferencia de conocimiento estratégico de solución de problemas en estudiantes en etapas iniciales de aprendizaje de la programación. Las teorías mencionadas arriba se toman como referencia para proponer los principios de diseño de tal clase de artefacto de software.

Se desarrolló y probó experimentalmente un prototipo de sistema visor de protocolos aplicando la teoría de diseño de sistemas de información propuesta, con alumnos de primer año de Ingeniería en Sistemas Computacionales e Ingeniería en Electrónica de la Universidad Autónoma de Aguascalientes, obteniéndose resultados estadísticamente favorables (ANOVA, estadística descriptiva, análisis de correlación) que sugieren que el artefacto puede ayudar a mejorar el aprendizaje de la programación en alumnos que inician el estudio de esta disciplina.

INDICE DE CONTENIDO

INTRODUCCIÓN **17**

1 PROBLEMA DE INVESTIGACIÓN **19**
1.1 Antecedentes sobre el aprendizaje. 19
1.2 Problema de investigación 22
1.3 Revisión de literatura. 24
1.3.1 Los modelos mentales. 25
1.3.2 Programadores expertos y novatos. 28
1.3.3 Las estrategias, esquemas o planes. 30
1.3.4 La Metacognición. 33
1.3.5 Otros factores cognitivos. 36
1.3.6 La enseñanza tradicional de la programación. 39
1.3.7 Herramientas para aprender a programar 40
1.4 Justificación. 47
1.5 Objetivos. 49
1.6 Hipótesis 52

2 TEORÍAS DE DISEÑO DE SISTEMAS DE INFORMACIÓN (ISDT). **54**
2.1 Componentes de una ISDT. 55
2.2 Planteamiento de una ISDT. 58

3 TEORÍA DE DISEÑO DE SISTEMAS DE INFORMACIÓN PARA SISTEMAS VISORES DE PROTOCOLOS VERBALES. **62**
3.1 Producto de diseño. 62
3.1.1 Teorías núcleo. 63
3.1.1.1 Teoría de Codificación Dual. 65
3.1.1.2 Protocolos verbales 69
3.1.1.3 Efecto del ejemplo resuelto y el problema por completar. 73
3.1.2 Metarequerimientos 78
3.1.3 Meta Diseño 83
3.1.4 Hipótesis del producto de diseño 86
3.2 Proceso de diseño. 87
3.2.1 Método de diseño 87
3.2.2 Teorías núcleo. 88
3.2.2.1 Prototipos evolutivos incrementales. 88
3.2.3 Hipótesis del proceso de diseño 93

4 DESARROLLO DE UN SISTEMA VISOR DE PROTOCOLOS VERBALES **95**
4.1 Desarrollo del artefacto. 96
4.1.1 Arquitectura 96
4.1.2 Modelo de datos 99
4.1.3 Interfaz de usuario. 105
4.2 Evaluación del artefacto. 111
4.2.1 Prueba de usabilidad. 113
4.2.1.1 Prueba piloto. 114
4.2.1.2 Prueba formal. 116

4.2.2 Modelo de investigación. 122
4.2.3 Prueba experimental. 123
4.2.3.1 Instrumento de medición. 124
4.2.3.2 Condiciones del estudio experimental. 126

5 RESULTADOS. 130
5.1 Análisis de varianza (ANOVA) 131
5.2 Estadística descriptiva. 132
5.3 Distribución de frecuencias. 133
5.4 Análisis de correlación 136
5.5 Resultados finales 137

6 CONCLUSIONES. 139
6.1 De los objetivos e hipótesis del estudio. 139
6.2 De la aplicación de la metodología. 143
6.2.1 De las hipótesis de producto y proceso de diseño. 144
6.3 Limitaciones del estudio. 147
6.4 Aportaciones del estudio. 148
6.5 Recomendaciones para trabajos futuros. 151

ANEXOS

ANEXO I. La naturaleza de las teorías de diseño. 154
ANEXO II. Tareas y cuestionario para prueba de usabilidad. 156
ANEXO III. Instrumento de medición 161
ANEXO IV. Problemas cargados en el prototipo del Sistema Visor de Protocolos (SVP) .162
ANEXO V. Retroalimentación cualitativa de prueba de usabilidad. 163
ANEXO VI. Publicaciones generadas. 165

7 GLOSARIO. 194

8 BIBLIOGRAFIA 197

ÍNDICE DE FIGURAS

Figura 1. El aprendizaje visto como un continuo. 20
Figura 2. Sistemas para enseñar a programar. 41
Figura 3. Sistemas para fomentar la programación y desarrollar la lógica. 43
Figura 4. Relación entre componentes de una ISDT. 58
Figura 5. Planteamiento de una teoría de diseño de sistemas de información (ISDT) 60
Figura 6. Planteamiento de una ISDT para un VIS. 61
Figura 7. Planteamiento de una ISDT para un SVP 62
Figura 8. Los sistemas de memoria de la mente humana 63
Figura 9. Modelo de la memoria de trabajo. 64
Figura 10. Teoría de la codificación dual. 66
Figura 11. Proceso de construcción de prototipos evolutivos e incrementales 89
Figura 12. Arquitectura funcional del artefacto visor de protocolos verbales 97
Figura 13. Perspectiva arquitectónica complementaria. 98
Figura 14. Modelo E-R propuesto para la instanciación del artefacto. 104
Figura 15. Primera versión de interfaz de usuario del visor de protocolos 108
Figura 16. Interfaz de usuario prototipo en una herramienta 4GL (Access) 109
Figura 17. Interfaz de usuario del prototipo basada en Web 110
Figura 18. Ejemplo de prueba piloto mediante protocolo de pensamiento en voz alta. 115
Figura 19. Opción de ir a un paso específico del protocolo verbal. 119
Figura 20. Identificar el punto focal de un programa en el prototipo visor de protocolos. 120
Figura 21. Opción de búsqueda específica de protocolos 121
Figura 22. Modelo de investigación. 122
Figura 23. Histograma de frecuencias de la prueba piloto. 126
Figura 24. Histograma grupo EXP. 135
Figura 25. Histograma grupo TRAD. 135
Figura 26. Ejemplo de tutor cognitivo. 152

ÍNDICE DE TABLAS

Tabla 1. Justificación de la investigación. 48
Tabla 2. Componentes de una Teoría de Diseño de Sistemas de Información 57
Tabla 3. Relación ortogonal entre los códigos mentales y los cinco sentidos en la DCT 67
Tabla 4. Proposiciones de Codificación Dual 80
Tabla 5. Proposiciones sobre protocolos verbales 81
Tabla 6. Proposiciones sobre ejemplos resueltos y problemas por completar. 82
Tabla 7. Meta-requerimientos de un SVP 83
Tabla 8. Meta-diseño de un SVP 84
Tabla 9. Hipótesis comprobables del producto de diseño. 86
Tabla 10. Selección de la estrategia de prototipeo adecuada 92
Tabla 11. Hipótesis de proceso de diseño de un SVP. 93
Tabla 12. Transcripción literal de un protocolo verbal 100
Tabla 13. Ejemplo de un protocolo verbal segmentado. 102
Tabla 14. Métodos de evaluación de artefactos.. 111
Tabla 15. Versión preliminar de lista de tareas para prueba de usabilidad. 114
Tabla 16. Tareas para prueba de usabilidad a partir de prueba piloto. 116
Tabla 17. Resultados del cuestionario de la prueba de usabilidad. 117
Tabla 18. Retroalimentación cualitativa de la prueba de usabilidad 118
Tabla 19. Descripción de variables de modelo de investigación. 123
Tabla 20. Estadística descriptiva del instrumento de medición. 125
Tabla 21. Criterios de calificación del instrumento de medición. 130
Tabla 22. Resultados prueba ANOVA entre grupo experimental y de control. 132
Tabla 23. Estadística descriptiva de prueba experimental. 133
Tabla 24. Tabla de frecuencias de calificaciones. Grupo experimental 133
Tabla 25. Tabla de frecuencias de calificaciones. Grupo de control. 134
Tabla 26. Correlación entre experiencia previa y aprendizaje de la programación. Grupo EXP. 136
Tabla 27. Correlación entre experiencia previa y aprendizaje de la programación. Grupo TRAD. 137
Tabla 28. Síntesis de hipótesis de producto y proceso de diseño y resultados obtenidos. 145

INTRODUCCIÓN

La estructura de la presente tesis se divide en los capítulos que se sintetizan a continuación.

En el capítulo uno se dan antecedentes sobre la naturaleza del aprendizaje y se delimita que el problema de investigación se circunscribe al aprendizaje de la programación básica. Se sintetizan los hallazgos reportados por la literatura en los últimos treinta años asociados a la psicología de la programación y se señalan los aspectos críticos del fenómeno. Se describen las características de los cursos tradicionales de programación y se hace un resumen de los tipos de herramientas de software para apoyar el aprendizaje de la programación que se han desarrollado en años recientes. Se plantean además los objetivos, justificación e hipótesis de la investigación.

En el capítulo dos se describen los antecedentes, naturaleza y componentes de las Teorías de Diseño de Sistemas de Información (Information Systems Design Theory o ISDT, por sus siglas en inglés), que se usó como marco metodológico de referencia para alcanzar los objetivos de la tesis.

En el capítulo tres, se plantea una Teoría de Diseño para crear una clase de artefacto de software denominada Sistema Visor de Protocolos Verbales, cuyo objetivo es fomentar la transferencia de conocimiento estratégico y aprendizaje significativo para la solución de problemas básicos de programación, identificando Meta-requerimientos, Teorías Núcleo, Meta Diseño e Hipótesis comprobables de producto y proceso de diseño, de acuerdo con los principios propuestos en el capítulo dos.

En el capitulo cuatro se describe el proceso de desarrollo y prueba de un artefacto de software teniendo en cuenta el Meta-diseño descrito en el ISDT para Sistemas Visores de Protocolos descrito en el capítulo tres. Se muestra un proceso de prueba de usabilidad de la interfaz de usuario del artefacto y las condiciones de la prueba experimental, así como el modelo de investigación utilizado.

En el capítulo cinco se muestran los resultados estadísticos de la prueba experimental, en donde se analizan los datos arrojados por ANOVA, la estadística descriptiva de los grupos participantes y el análisis de correlación entre experiencia previa y aprendizaje.

El capítulo seis muestra las conclusiones de la tesis, en donde se revisan los objetivos de la tesis y se verifica la hipótesis de investigación. Se describe la experiencia de aplicar el marco ISDT como guía metodológica, se contrastan las hipótesis de producto y proceso de diseño, se señalan las limitaciones del estudio, se enumeran sus aportaciones y se siguieren futuras líneas de investigación.

1 PROBLEMA DE INVESTIGACIÓN

1.1 Antecedentes sobre el aprendizaje.

La actividad creativa puede describirse como un tipo de proceso de aprendizaje en donde el maestro y el alumno están ubicados en el mismo individuo.

- Arthur Koestler, novelista y periodista (1905-1983)

El aprendizaje consiste en la adquisición de conocimiento. El conocimiento, en la vida cotidiana, se manifiesta en quien lo posee por su capacidad para resolver problemas. En la literatura de investigación educativa es frecuente la distinción entre el aprendizaje superficial (o rutinario, o memorístico) y el aprendizaje significativo (o profundo) (Ausubel 1963; Joseph D. Novak y Alberto J. Cañas 2006; Roll et al. 2007). El aprendizaje significativo existe cuando una persona cuenta con los conocimientos necesarios para resolver problemas exitosamente.

La resolución de problemas involucra la definición de estrategias para alcanzar un objetivo que uno nunca ha alcanzado previamente. Es decir, consiste en encontrar la manera de cambiar una situación de un estado inicial a un estado deseado (Mayer 1992). Cuando esto sucede (cuando una persona logra resolver problemas a través de un aprendizaje significativo) se dice que tuvo lugar una *transferencia de conocimiento*.

Así, la transferencia tiene lugar cuando se usa lo aprendido para resolver nuevos problemas, contestar nuevas preguntas o facilitar el aprendizaje de nuevos temas (Mayer y Wittrock 1996).

Por contraste, el aprendizaje superficial o memorístico, es aquel que sucede cuando almacenamos conceptos y estructuras cognitivas que no están conectadas a la solución de problemas. Algunos autores llaman conocimiento inerte a aquel que se adquiere por medio del aprendizaje superficial (Bereiter y Scarmadalia 1985).

De acuerdo con este enfoque, un alumno con aprendizaje superficial, podría retener el suficiente conocimiento para aprobar un examen, pero carecería de la capacidad de aplicarlo a situaciones nuevas y diferentes a las memorizadas.

Novak y Cañas (2006) aclaran que la distinción superficial/significativo no es en realidad una dicotomía, sino los extremos de un mismo continuo (Figura 1) en donde la creatividad puede verse como un nivel muy alto del aprendizaje significativo.

Figura 1. El aprendizaje visto como un continuo.

El aprendizaje superficial parece funcionar bien cuando el tema de estudio forma un cuerpo estático de conocimiento (tal como la historia o la geografía) en donde llega primero la memorización de fechas, y el análisis y la comprensión llega después; pero no funciona en dominios como la programación de computadoras, la cual más que un conjunto de conocimientos, es realidad un conjunto de habilidades (Jenkins 2002).

En vista de lo anterior, en el contexto del aprendizaje de la programación, en primera instancia puede creerse que es indispensable un aprendizaje profundo para poder adquirir las habilidades de solución de problemas requeridas por este dominio. Sin embargo, también puede argumentarse que la programación se aprende como un simple proceso de ajuste de patrones, en donde se detectan problemas comunes y se aplican ciertos tipos de soluciones y estrategias estereotipadas. Bajo este segundo enfoque se pensaría que se habla nuevamente de aprendizaje superficial.

Es así que la naturaleza de la programación se encuentra de hecho en un punto intermedio entre estos dos tipos de aprendizaje (Du Boulay 1989; Milne y Rowe 2002). Por ejemplo: el aprendizaje superficial puede ser útil para aprender detalles de sintaxis o precedencia de operadores, pero también se requiere un aprendizaje profundo si es que se desea alcanzar una verdadera competencia. Aquí radica el punto clave: ambos tipos de aprendizaje deben aplicarse *al mismo tiempo*. Es decir, en programación no tiene sentido memorizar primero las reglas de sintaxis de algún lenguaje de programación y después avanzar a escribir un programa. Así, los cursos tradicionales suelen ser razonablemente efectivos en la enseñanza de los elementos que requieren aprendizaje memorístico, pero suele presentar mucha mayor dificultad transferir a los alumnos habilidades que en principio requieren de aprendizaje significativo.

Esta situación pone a la programación más allá de la experiencia educativa de la mayoría de los estudiantes[1].

[1] Existen, por supuesto, otros enfoques sobre el tema del tipo o estilo de aprendizaje, además de la distinción entre aprendizaje superficial y significativo mencionado aquí. Ver por ejemplo (Kolb 1984; Felder 1996).

1.2 Problema de investigación

La presente tesis se circunscribe al estudio del problema del aprendizaje de la programación básica, en el contexto de cursos introductorios de nivel universitario en donde estudiantes con distintos antecedentes de programación y perfil, intentan adquirir la habilidad de escribir programas de computadora, en carreras afines a las ciencias computacionales.

En lo que respecta a la disyuntiva –aún en debate– que existe en el diseño curricular sobre cual paradigma de programación (programación estructurada o programación orientada a objetos) enseñar primero (Kölling 1999; Avello 2003; C. Hu 2004; Sajaniemi 2006; Pont 1998), aquí se toma la postura de estudiar el problema del aprendizaje de la programación básica sobre el paradigma estructurado o procedural.

El problema del aprendizaje de la programación se ha estudiado desde múltiples ángulos. En esta tesis, dada la complejidad del tema, se toca el problema (a distintos grados de detalle), desde las siguientes perspectivas:

a) Desde el punto de vista **cognitivo** en cual se ha entendido que el proceso de programar es básicamente un proceso mental, y en donde puede encontrarse gran cantidad de literatura que describe las características de los procesos mentales implicados en la programación. Se hace una revisión detallada de los aspectos cognitivos del aprendizaje de la programación en las secciones 1.3.1. a 1.3.5.

b) Desde un enfoque de **aplicaciones de software**, en donde se han desarrollado de años a la fecha, diversas herramientas cuyo objetivo es ayudar

en el aprendizaje de la programación con distintos grados de éxito (ver sección 1.3.7). Implícitamente, este enfoque parte de la premisa de que el rendimiento en el aprendizaje en general puede mejorarse si el estudiante es apoyado por aplicaciones de software que provean mecanismos de naturaleza audiovisual, interactiva y/o de inteligencia artificial. En la presente tesis, se parte de este principio y se toma como punto de partida para proponer una clase de artefacto de software que aporte conocimiento para la solución del problema.

c) La propia **complejidad** del dominio de la programación, en donde se sabe que el acto de programar es inherentemente complejo, dado que demanda del estudiante (como se ha mencionado en la sección anterior) la aplicación simultánea de diversas habilidades, múltiples procesos, lenguajes de programación con distintas características sintácticas y distintos estilos y ritmos de aprendizaje.

d) El enfoque de la **enseñanza tradicional** de la programación (ver sección 1.3.6) en donde se reporta que la enseñanza se basa casi exclusivamente en temas sintácticos, tales como el significado de las instrucciones de un lenguaje de programación, las reglas de precedencia de los operadores o los tipos de datos.

Otras perspectivas del problema que también han sido exploradas, ven el fenómeno desde un **punto de vista social** (Booth 2001; Bruce 2004) en donde se analiza al estudiante como parte de un proceso de “enculturación” (es decir, aprender a programar incluye volverse parte de la cultura de los programadores), pero este enfoque no está contemplado en esta tesis.

1.3 Revisión de literatura.

Los buenos programadores no temen usar sus cerebros, pero hay guías que nos ayudan a no tener que pensar todos y cada uno de los casos.

-Francis Glassborow

El acto de programar es esencialmente un proceso cognitivo de resolución de problemas, que deriva en la escritura de representaciones abstractas de un proceso algorítmico. En otras palabras, programar consiste en idear mentalmente un proceso de solución a un problema, combinando simultáneamente un conjunto de estructuras sintácticas predefinidas, por medio de un lenguaje de programación. Es así que la mayor parte de la literatura relacionada con el problema del aprendizaje de la programación se encuentra en el ámbito de la psicología cognitiva.

El concepto de psicología cognitiva fue descrito por Urlic Neisser (Neisser 1967), quien sugirió que las personas son como "sistemas de procesamiento de información dinámicos, cuyas operaciones mentales pueden ser descritas en términos computacionales". Así, la psicología cognitiva estudia los procesos mentales internos tales como el procesamiento visual, la memoria, la resolución de problemas y el lenguaje.

La literatura cognitiva sobre el aprendizaje de la programación es extensa y compleja. El tema fue de especial interés durante los años 80s (J. Anderson, Farrell, y Sauers 1984; Bayman 1983; Larkin y Simon 1987) y en años recientes continúa siendo explorado (Ma et al. 2007; Ramalingam 2004; Wiedenbeck, LaBelle, y Kain 2004); de tal manera que a la fecha continúan identificándose aspectos que influyen (o no) en el fenómeno. Los modelos mentales viables, las diferencias en el desempeño de expertos y novatos, la existencia de las llamadas estrategias o planes, Recientemente, la llamada Metacognición (entendida como

la autoconciencia del conocimiento y procesos cognitivos que tiene la persona) son algunos de los hallazgos que se han identificado como críticos en la literatura cognitiva del aprendizaje de la programación de los años últimos.

1.3.1 Los modelos mentales.

Cuando inicias el studio de una disciplina, parece como que si tuvieras que memorizar un millón de cosas. No tienes que hacerlo. Lo que debes hacer es identificar los principios básicos (generalmenre entre tres y doce de ellos) que gobiernan el campo. El millón de cosas que creías que tenías que memorias, son simples variaciones de esos principios básicos.

-John T. Reed.

Norman (1983) define un modelo mental como "la conceptualización que un usuario tiene de un sistema objetivo". Por su parte, George (2000) señala que un modelo mental es la representación interna de una tarea o sistema complejo, cuya construcción permite al aprendiz el razonar, predecir y comprender el funcionamiento de tal tarea o sistema. Norman (1983) indica además que un modelo mental se desarrolla con el tiempo como resultado de la interacción que tenga el sujeto con el sistema objetivo y que no es indispensable que tal modelo mental sea muy preciso, pero si "funcional". Los factores que afectan el desarrollo de un modelo mental incluyen el conocimiento técnico precedente del usuario, la experiencia previa con sistemas similares y la propia estructura de patrones de pensamiento del usuario.

Se aclara además que las personas tienen una tendencia a desarrollar estrategias generales que parecen ajustarse en principio a todos los sistemas. Así, el modelo mental sirve como punto de referencia: el usuario "ejecuta" el modelo para predecir el resultado de la operación de un sistema.

En programación, un modelo mental se refiere a la imagen que tiene el usuario sobre el procesamiento invisible que ocurre dentro de la máquina entre una entrada y una salida (Bayman 1983). Johnson-Laird (1983) indica que escribir un programa involucra mantener muchos y diversos modelos mentales, muy separados de lo que es el modelo sintáctico propio del lenguaje de programación.

En los estudios cognitivos sobre el aprendizaje de la programación, la evaluación de los modelos mentales de los estudiantes se hace por medio de exámenes de comprensión de código. En un ejemplo tomado de Dehnadi y Bornat (2006) a un programador novato se le provee el siguiente fragmento de código:

```
int a = 10;
int b = 20;
a = b;
```

Posteriormente se le cuestiona sobre los valores finales de a y b. Si su respuesta es correcta, se infiere que su modelo mental también es correcto.

Winslow (1996, p.21), advierte que los modelos mentales son críticos para el aprendizaje de la programación: modelos sobre estructuras de control, estructuras de datos, diseño de programas y dominio del problema, son muy importantes. Señala además que si el instructor omite estos modelos, los estudiantes crearán de cualquier modo los suyos propios, de calidad dudosa.

Por su parte, Du Boulay (1989) definió el concepto –frecuentemente referenciado– de *maquina nocional*, que se entiende como el modelo mental que tiene el usuario sobre la computadora cuando ésta ejecuta programas, y que le sirve para contestar la pregunta “¿qué tipo de comandos son los que entiende la computadora?”

Esta máquina nocional se crea respecto al lenguaje de programación: es decir, la que subyace al lenguaje Pascal, sería diferente de la del lenguaje Prolog.

Du Boulay (1989, pp. 297-298) continúa indicando que “un punto importante es la necesidad de proveer al principiante con algún modelo o descripción de la máquina que está operando por medio del lenguaje de programación[2], pero advierte que “inventar una historia consistente no siempre es sencillo”.

Dados los argumentos anteriores, se entiende la necesidad de que al momento de la instrucción al programador novato le sean provistos explícitamente estos modelos mentales válidos sobre el funcionamiento de secuencias de código de un programa.

Sin embargo la literatura cognitiva aporta pocas guías sobre como fomentar estos modelos mentales válidos en un contexto instruccional. Es decir, se enfoca en el diseño de instrumentos de medición y el análisis de los resultados obtenidos, pero es difícil encontrar ejemplos concretos sobre cómo aplicarlos efectivamente como parte del contenido didáctico de una materia de introducción a la programación.

[2] Algunos instructores señalan, anecdóticamente, que el uso del “Debugger” de algunos entornos de programación, suele proporcionar al estudiante esta imagen mental del funcionamiento interno del lenguaje de programación.

1.3.2 Programadores expertos y novatos.

No les puedes enseñar a los programadores principiantes programación top-down, porque no saben cual lado es el de arriba.

-C.A.R. Hoare

Rist (2004, p.176) advierte que "la destreza en la resolución de problemas se logra resolviendo muchos, muchos problemas de diferentes tipos y tamaños, a lo largo del tiempo". Aclara que los libros pueden proveer hechos, principios y ejemplos, pero que estas piezas sueltas "deben conectarse durante el proceso de resolución de problemas".

Las investigaciones que comparan el desempeño de programadores expertos y novatos frecuentemente buscan diferencias entre sus modelos mentales para buscar acelerar el proceso de aprendizaje e identificar errores en las estrategias de resolución de problemas de los aprendices (Fixx, Wiedenbeck, y Scholtz 1993; Hegarty 1993; Ma et al. 2007; Ramalingam 2004).

Por ejemplo, estudios sobre el comportamiento de expertos se enfocan en analizar cómo éstos son capaces de crear sofisticadas representaciones de conocimiento y aplicar estrategias de resolución de problemas. Von Mayrhauser y Vans (1994), hacen un resumen de las características que los programadores expertos tienen en común:

a) tienen esquemas bien organizados de conocimiento especializado;
b) organizan su conocimiento de acuerdo a características funcionales (tales como la naturaleza del algoritmo subyacente, en lugar de detalles superficiales de sintaxis);
c) usan estrategias de resolución de problemas tanto generales (como divide y vencerás) como especializadas;

d) son flexibles en sus estrategias para comprender problemas y pueden abandonar fácilmente hipótesis que resultan falsas; y
e) usan estrategias “de arriba hacia abajo” para descomponer y comprender programas.

Jenkins (2002) señala que programar no es en realidad una sola habilidad, sino un conjunto de habilidades que forman una jerarquía. De acuerdo con esta proposición, un estudiante que intenta aprender a programar debe comenzar con las habilidades del nivel más bajo en la jerarquía para posteriormente progresar “hacia arriba”[3]. Fixx et al. (1993), señalan que una diferencia entre los modelos mentales de expertos y novatos, es el reconocimiento en los primeros de un patrón “jerárquico y multicapa”, que subyace en el texto de un programa.

Robins, et.al. (2003) señalan que muchas las características de programadores expertos son compartidas por todos los expertos en general, como se ha explorado en otros campos, tales como las matemáticas. Mencionan que los expertos son buenos para reconocer, usar y adaptar patrones de solución.

Por contraste, se infiere que estas características no están presentes en los programadores novatos. Así, Winslow (1996) concluye que los programadores novatos:

a) están limitados a conocimiento superficial,
b) carecen de modelos mentales detallados,
c) su enfoque hacia la programación es “línea por línea”, en lugar de utilizar bloques o estructuras significativas.

[3] Mientras que la noción de una estructura jerárquica de habilidades es frecuentemente citada, algunos estudios han mostrado que alumnos que parecen haber dominado una serie de habilidades pre-requisito, son incapaces de avanzar a un siguiente nivel jerárquico (Mayer 1998). Es decir, poseer las habilidades básicas es necesario, pero no suficiente. Procesos cognitivos tales como la metacognición parecen influir.

Por ejemplo, Ben-Ari (1998), indica que el concepto de variable es extremadamente difícil para los programadores novatos y Dehnadi y Bornat (2006, 5) indican que "La operación de asignación y el concepto de secuencia dan la impresión de no ser problemáticos para los programadores novatos, pero en realidad son un verdadero obstáculo, que se encuentra al propio inicio de la mayoría de los cursos de programación". Ilustran, por ejemplo, que algunos estudiantes novatos creen que al ejecutar la instrucción A = B, la variable B deja de contener un valor. Aquí el estudiante construye un modelo "consistente" usando la analogía de una caja que se queda vacía, pero sucede que este modelo resulta ser no viable.

Es así que la evidencia proporcionada por esta línea de investigación refuerza la conclusión de la sección anterior, en donde se sugiere la necesidad de evitar que el estudiante novato cree modelos mentales intuitivos por analogía (tomados de su experiencia previa del mundo real) y que modelos viables deben ser explícitamente enseñados, mostrando el funcionamiento interno correcto de la computadora al ejecutar las instrucciones de un programa.

1.3.3 Las estrategias, esquemas o planes.

Primero resuelve el problema. Después, escribe el código.
-John Johnson.

Relacionado con las diferencias entre programadores novatos y expertos, se ha encontrado que un componente faltante en el modelo mental de un novato es la falta de estrategias y soluciones predefinidas (llamados también esquemas, planes o clichés) a tipos de problemas de programación recurrentes, así como su incapacidad para reconocer dichos tipos de problemas (Fixx, Wiedenbeck, y Scholtz 1993).

Brooks (1990), señala que la aplicación de una estrategia impacta fuertemente el diseño del programa final elaborado, y que estas estrategias no pueden (en la mayoría de los casos) ser deducidas de la forma final del programa. Es decir, que el estudio del programa terminado puede dar mucha información al programador novato sobre conceptos de sintaxis y características del lenguaje, pero no sobre las estrategias que se usaron para escribirlo. Estas estrategias son mucho más difíciles de hacer explícitas en el proceso de enseñanza.

De acuerdo con Soloway y Ehrlich (1984) los planes, sirven para el manejo de situaciones estereotípicas que surgen frecuentemente durante la escritura de un programa.

Al respecto Rist (1995, p.514) comenta que "en el estudio de la programación existe considerable evidencia empírica acerca de que la estrategia es el componente cognitivo básico usado en el diseño y comprensión de programas". Una estrategia es una solución abstracta a un problema de programación común. Es abstracta porque puede ser usada en muchas situaciones. Estos esquemas (o planes) pueden acoplarse entre sí, permitiendo que esquemas más pequeños conformen planes más grandes (Rist 2004, p.175).

Richard Brooks (Brooks 1977; Brooks 1983) describe a la comprensión de programas como un proceso "de arriba hacia abajo, guiado por hipótesis". Sugirió que en lugar de estudiar programas línea por línea, los programadores expertos forman hipótesis con base en su conocimiento del dominio de la programación, y que éstas son verificadas o falsificadas al encontrarse "llaves" o "faros[4]", en el código del programa, que indican la presencia de estructuras o funciones características específicas.

[4] En inglés el término se conoce como *Beacon.*

Este hallazgo es retomado por Rist (1995, p.537) quien lo rebautiza, proponiendo el concepto de “foco”, el cual puede consistir ya sea de una sola línea, o de un bloque de código, que representa *la operación principal* en un programa. El foco es a su vez el punto de partida para el *diseño focal* de programas, que ocurre cuando éste se descompone y se identifica la acción u objeto más básico, construyéndose el resto de la solución alrededor de dicho foco.

A su vez, un protocolo de diseño (que sirve para registrar el proceso de solución de un problema de programación) registra el orden en el que las piezas de conocimiento se combinan para crear una solución de software (Rist 2004).

Robins et.al. (2003,p.166) sugieren, que “las estrategias de programación deben recibir más atención explícita en los cursos introductorios” y que una forma de hacerlo es “introducir muchos ejemplos de programas tal y como están siendo desarrollados y discutir las estrategias usadas”. Spohrer y Soloway (1989), comentan que a los estudiantes de programación no se les da suficiente instrucción sobre como “juntar las piezas” de un programa y que al enfocarse explícitamente en las estrategias se puede ayudar a disminuir esta tendencia.

Davies (1993, p.265) señala que el proceso de escritura de programas no debe entenderse como la “transcripción literal de una representación internamente almacenada y terminada”, sino realmente como un proceso exploratorio e incremental, determinado por episodios menores de resolución de problemas y una re-evaluación frecuente del problema principal.

Por lo tanto, la evidencia indica que la existencia de estrategias efectivas es crucial para el desarrollo de un programa, entendiendo por estrategia o plan a la selección, aplicación y constante revisión de un conjunto de soluciones pre-existentes en la estructura de conocimiento un programador, cuya efectividad es corroborada o rechazada por el propio programador durante el proceso mismo de

construcción del programa, conforme se avanza en el logro del objetivo que marca la solución del problema.

1.3.4 La Metacognición.

Otra técnica efectiva es explicar tu código a alguien más. Esto frecuentemente ocasionará que te expliques el error a ti mismo. Algunas veces, no toma más que unas cuantas frases, seguidas de un "Ah, olvídalo, ya veo lo que está mal, disculpa que te haya molestado". Esto funciona extremadamente bien; uno puede incluso usar como oyentes a personas que no saben programar.

Una facultad de informática en una universidad mantenía un osito de peluche en una mesa de ayuda. A los estudiantes con errores misteriosos en sus programas, se les solicitaba que se lo explicaran al oso, antes de hablar con un asesor humano.

-Brian Kernighan y Rob Pike, sobre la detección y corrección de errores.

Flavell (1979) describió a la Metacognición como la conciencia sobre como la propia persona aprende. Es decir: mientras que las estrategias cognitivas le permiten a la persona construir conocimiento o solucionar problemas, La metacognición le permite monitorear y mejorar su progreso, evaluar su comprensión y aplicar el conocimiento a nuevas situaciones.

Gourgey (1998) señala que a través de la metacognición el individuo puede definir la naturaleza de un problema o tarea, seleccionar una representación física o mental útil, seleccionar la estrategia más efectiva para ejecutarla, activar conocimiento previo relevante, prestar atención a la retroalimentación sobre cómo se está avanzando en la resolución del problema o tarea, y traducir esta retroalimentación para mejorar el desempeño si es necesario, ya sea durante la ejecución de la tarea o en una situación futura.

Por ejemplo, en el ámbito del aprendizaje de las matemáticas, Schoenfeld (1987) encontró que los estudiantes novatos tendían a seleccionar rápidamente una estrategia de solución a un problema y se tomaban mucho tiempo ejecutándola, deteniéndose rara vez a evaluar su trabajo para ver si avanzaban hacia el objetivo. Estos aprendices al no contar con auto-monitoreo y auto-regulación, gastaban mucho de su tiempo en aplicar soluciones extravagantes que los llevaban en la dirección equivocada; y aún cuando contaban con el conocimiento matemático adecuado para resolver el problema, eran incapaces de activarlo constructivamente. En contraste, los matemáticos con experiencia se tomaban el tiempo para analizar el problema, asegurándose primero de comprenderlo, intentando muchas estrategias para ver si éstas funcionaban y cambiándolas inmediatamente si no lo hacían.

Lo anterior sugiere que un estudiante puede contar con todo el conocimiento y habilidades que son pre-requisito para resolver un problema, pero puede fallar al tratar de aplicarlos, por no saber cómo monitorear y evaluar las decisiones que ha tomado.

En el ámbito del problema del aprendizaje de la programación Williams y Upchurch (2001) reportan resultados favorables con la estrategia de enseñanza conocida como “programación por pares” (pair programming) en la cual un estudiante monitorea el avance y retroalimenta directamente al otro acerca de las decisiones, resultados y estrategias tomadas al momento de construir un programa. En este estudio, Williams y Upchurch (2001) señalan que mediante el mecanismo de programación por pares, el conocimiento declarativo, procedural y metacognitivo se intercambia constantemente entre los compañeros. Indican además que la adquisición de habilidades metacognitivas rara vez es considerada en la instrucción, siendo que la evidencia tanto en el ámbito de la programación como en otros dominios, indican que tales formas de conocimiento deben también ser explícitamente aprendidos por el estudiante.

En cursos introductorios de programación, otros investigadores (Arshad 2009) han aplicado técnicas pedagógicas con resultados favorables también directamente relacionadas con aspectos metacognitivos, tales como “decir en voz alta” el proceso de pensamiento de un programador con experiencia al resolver problemas básicos, para que estudiantes novatos tengan en cuenta las estrategias seguidas por éstos.

Sin embargo, el papel y alcance de la metacognición ha sido acotada por algunos investigadores. Por ejemplo Agre (1997) señala que muchas de las actividades cotidianas que hacemos (por ejemplo, manejar un automóvil, conversar) las hacemos sin utilizar la metacognición. Roth (2004) advierte que la metacognición de hecho puede afectar el rendimiento en una tarea, ya que consume recursos mentales adicionales, aún cuando este rendimiento sea de naturaleza no cognitiva.

Más importante aún, advierte Roth (2004), es la pregunta “¿Cómo podemos ser metacognitivos –es decir, revisar errores, monitorear tareas– cuando el dominio nos es poco familiar?” en cuyo caso, -continúa- en realidad hay muy poco que podamos hacer para evaluar si algunas de nuestras acciones son apropiadas o inapropiadas.

En otras palabras, a menos que los estudiantes posean por lo menos un *entendimiento conceptual rudimentario* del problema que están tratando de resolver, la actividad pedagógica –aún cuando incluya un enfoque en la metacognición– puede derivar en muy poco aprendizaje significativo en el estudiante.

Miles et al. (2003) enfatizan la importancia de que los estudiantes de programación adquieran “conocimiento estratégico”, el cual –indican– incluye habilidades metacognitivas tales como la prueba de programas, la depuración de errores de lógica y el monitoreo de la efectividad de las estructuras utilizadas.

Así, la evidencia señala que las estrategias aplicadas para resolver un problema de programación incluyen aspectos metacognitivos. Es decir, la metacognición guía la selección y aplicación de la estrategia. Esta capacidad metacognitiva, le permite al experto monitorear su progreso, reconocer patrones o tipos de problemas, aplicar otras estrategias y corregir decisiones de diseño equivocadas.

1.3.5 Otros factores cognitivos.

Como ya se ha dicho, la literatura que estudia el fenómeno del aprendizaje de la programación es extensa y diversa. En este sentido, investigadores han estudiado diversos factores relacionados con el fenómeno, además de los descritos en secciones anteriores.

Capacidad para las matemáticas. Algunos investigadores (P. Byrne 2001; Chenglie Hu 2006; White 2003) han señalado que la habilidad para las matemáticas tiene una fuerte correlación con el desempeño del estudiante durante el primer curso de programación. La programación requiere que el estudiante lleve a cabo procesos cognitivos, en los que el razonamiento abstracto es preponderante (Jenkins 2002). En este sentido, la similitud con las matemáticas es evidente. Pero la implicación de estas conclusiones, en un contexto de enseñanza, deriva en decisiones éticas delicadas. Por ejemplo ¿es deseable favorecer el ingreso a las carreras de ciencias computacionales solo a alumnos con mayor capacidad matemática? ¿Debe desalentarse el ingreso de aquellos alumnos con perfil matemático bajo? ¿Puede controlarse éste fenómeno en un entorno de educación superior?

Experiencia previa. Se ha reportado también que existe un efecto positivo entre la experiencia previa y el rendimiento académico en el primer curso de programación a nivel licenciatura (Fauxx 2006; Hagan y Markham 2000). Estos estudios enfatizan que la aplicación de cursos previos cuyo contenido sea orientado al desarrollo de habilidades de solución de problemas, desarrollo de algoritmos, pseudocódigo y técnicas de diagramación, antes de enseñar un lenguaje, tiene un efecto positivo en el aprendizaje de la programación, cuando el alumno llega al primer curso de programación a nivel licenciatura. Sin embargo, también se reporta que este efecto positivo solamente es perceptible durante dicho primer curso. El efecto tiende a desaparecer en los cursos subsecuentes, en donde el rendimiento de todos los participantes tiende a nivelarse (Holden y Weeden 2005).

Auto-eficacia. La auto-eficacia entendida como "lo bien que uno puede ejecutar cursos de acción para llevar a cabo situaciones prospectivas" (Bandura 1982, p.122) es un factor que se encuentra de manera recurrente en la literatura sobre psicología de la programación (Brosnan 1998; Ramalingam 2004; P. Byrne 2001). La autoeficacia se ha identificado como una influencia positiva en el rendimiento dentro de un primer curso de programación. Sin embargo, investigadores como (Heggestad 2005) han criticado la influencia de este factor, señalando que la auto-eficacia es realmente una variable residual del desempeño del participante en cursos anteriores.

El estilo de aprendizaje. En el ámbito de la enseñanza de la programación, se han realizado diversos estudios enfocados a medir el efecto del estilo de aprendizaje sobre el rendimiento académico. Byrne (2001) en un estudio realizado para medir el efecto del estilo de aprendizaje de los estudiantes en el rendimiento en un curso de programación, encontró que los estudiantes con un estilo convergente (Kolb 1984), tuvieron mayor éxito que aquellos con otros estilos de aprendizaje; pero advierte que estos resultados no son estadísticamente

significativos y que no indican que un estilo de aprendizaje sea predominante y que prediga éxito en los cursos de programación.

Por su parte Bishop-Clark (1995), al estudiar un grupo de estudiantes de primer curso, encontró que no había un claro efecto del estilo cognitivo y de la personalidad sobre el aprendizaje de la programación. Thomas et al. (2002), utilizando la escala de Felder (Felder 1996), encontraron que los estudiantes con estilo de aprendizaje tendiente hacia los extremos Reflexivo, Intuitivo, Verbal y Secuencial tenían mejores resultados que otros estilos. Pero nuevamente los autores concluyen que esto se atribuye más al sesgo del formato del contenido didáctico presentado en las materias de ingeniería, que la ventaja de un estilo de aprendizaje sobre otro.

En el análisis de estos factores adicionales, desde un punto de vista instruccional y práctico, se argumenta que la autoeficacia, la habilidad matemática, la experiencia previa y el estilo de aprendizaje, pueden ser difícilmente controlados por un instructor, enfrentado a la tarea de impartir un curso introductorio de programación, en el cual pueden existir alumnos de diferente perfil, experiencia, habilidad matemática, nivel de autopercepción y estilos de aprendizaje.

1.3.6 La enseñanza tradicional de la programación.

En la enseñanza de la programación, no se puede hacer que alguien sea un programador experto, de la misma manera en que estudiar pinceles y pigmentos no hacen a alguien un pintor experto.

-Eric Raymond

Un curso tradicional de programación se basa principalmente en sesiones en el aula y trabajo práctico en el laboratorio, en donde la mayoría del contenido se enfoca en las características del lenguaje que está siendo enseñado (Robins, J. Rountree, y N. Rountree, 2003).

Sobre este enfoque sintáctico de la programación, Brusilovsky (1998) indica, que los lenguajes de programación de uso general usados en los cursos introductorios (desde los lenguajes tradicionales de tercera generación, tales como pascal o C, hasta los lenguajes modernos orientados a objetos como Java) tienen la desventaja de proveer poco fundamento para explicar sus acciones básicas y estructuras de control, fomentando que el estudiante cree un modelo de "caja negra" sobre las entradas y salidas generadas por un programa en ejecución.

Linn y Clancy (1992); Linn (1985); McGill y Volet (1997); Volet y Lund (1994) concuerdan en que la mayoría de los cursos introductorios de programación son razonablemente buenos para enfatizar la comprensión de la sintaxis de los programas, pero que no se enfocan lo suficiente en el conocimiento estratégico requerido para escribir programas.

Una percepción común de los educadores en el área de la programación, consiste en asumir que tal conocimiento estratégico se desarrollará por sí mismo en el estudiante, como un subproducto de la propia currícula (Miles et al. 2003), en tanto que la literatura sugiere que éste debe ser explícitamente enseñado (A. Robins, J. Rountree, y N. Rountree 2003; Volet y Lund 1994).

1.3.7 Herramientas para aprender a programar

El uso de COBOL incapacita la mente; por lo tanto su enseñanza debería considerarse como una ofensa criminal.

-E.W. Dijkstra, científico y profesor de ciencias computacionales (1930-2002)

En las tres últimas décadas, investigadores preocupados por acercar la programación a un mayor número de personas, han desarrollado una gran variedad de aplicaciones orientadas a facilitar su aprendizaje. Desde Logo[5] (Papert 1980), hasta Alice (Conway 1997; 2002), y ya sea para fomentar la habilidad de resolución de problemas mediante el pensamiento lógico por medio de juegos, el diseño de interfaces de usuario alternativas o facilitando la transición hacia lenguajes de programación de propósito general, los diseñadores no han dejado de explorar nuevas formas de acercar la programación a las personas.

Kelleher y Pausch (2005) hicieron una revisión de aproximadamente 80 de estas aplicaciones y propusieron una taxonomía que dividieron en dos grandes grupos: a) sistemas para enseñar a programar y b) sistemas que fomentan el gusto por la programación.

La primera categoría (Figura 2) la conforman herramientas que proveen a los novatos una exposición temprana a los fundamentos del proceso de programar. Después de obtener experiencia con estos sistemas, se espera que los estudiantes avancen hacia lenguajes de propósito general, tales como Java, C o C++.

[5] Logo ha sido una aplicación muy influyente que ha inspirado muchas variaciones, tales como LogoBlocks, Leogo, Cleogo y Flogo, entre otras.

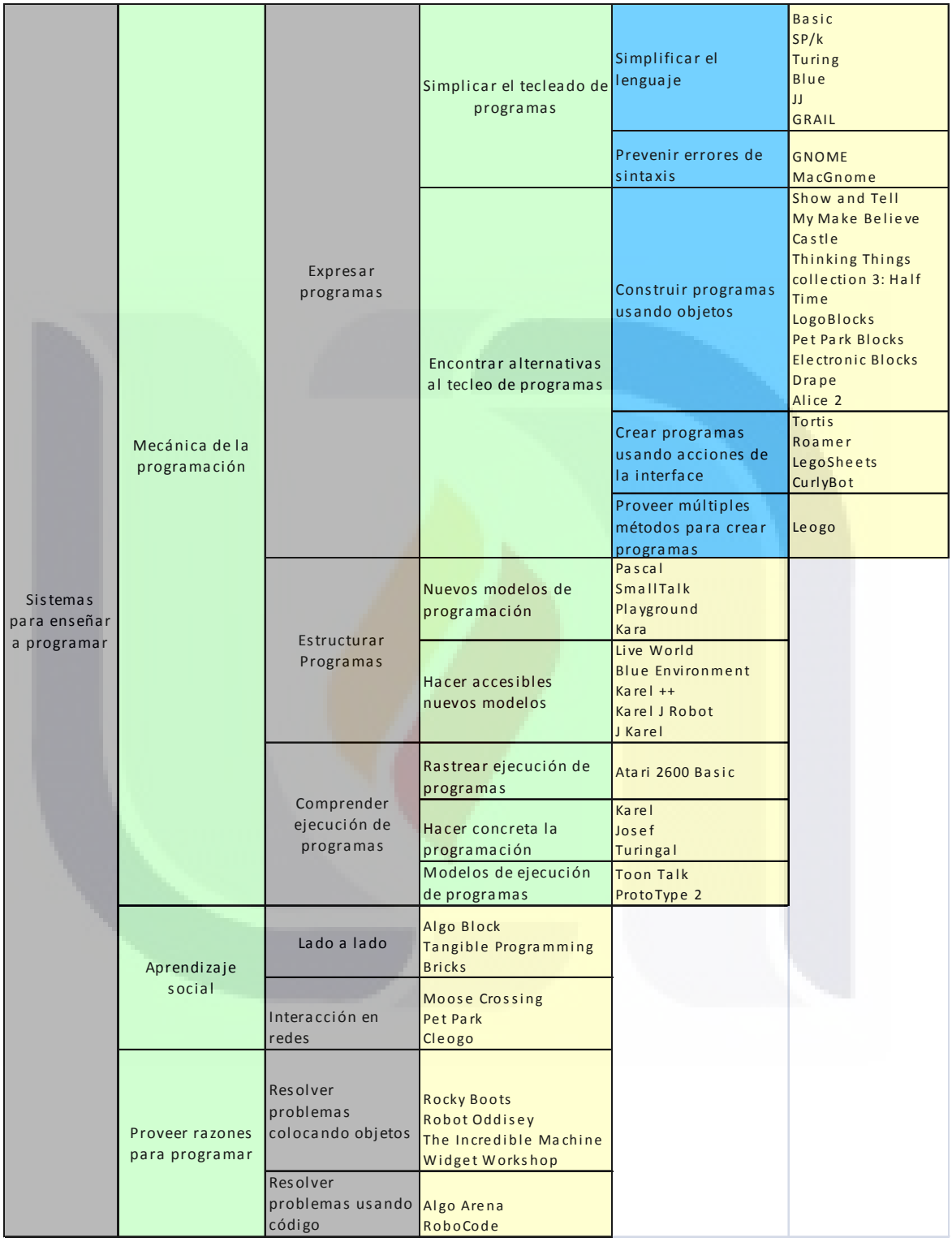


Figura 2. Sistemas para enseñar a programar.

Adaptado de (Kelleher y Pausch 2005).

Dentro de esta categoría Kelleher y Pausch proponen tres subconjuntos: las aplicaciones orientadas a facilitar la mecánica de la programación (¿Cómo expresar intensiones hacia la computadora? ¿Cómo comprender sus acciones?), las herramientas que fomentan un aprendizaje social (es más divertido y efectivo trabajar en grupo y proveer ejemplos y retroalimentación dentro de una comunidad) y las que brindan objetivos concretos para enfocar el esfuerzo de programación (es decir, proveer problemas de referencia para diferenciar la complejidad entre un proyecto y otro).

En la segunda categoría (Figura 3) existen sistemas diseñados con la creencia de que un aspecto importante de la programación es que esta "permite a las personas construir aplicaciones adaptadas a sus propias necesidades" (Kelleher y Pausch 2005, p.38), por lo que estos desarrolladores no están tan preocupados porque sus usuarios puedan trasladar el conocimiento hacia lenguajes de programación estándar, sino que puedan desarrollar un pensamiento lógico y una comprensión de cómo la programación puede resolver problemas cotidianos.

Una mención interesante es que los autores ubican al lenguaje de programación COBOL dentro de esta segunda categoría, tal vez como ejemplo de uno de los primeros intentos de hacer que los lenguajes de programación fuesen más comprensibles y cercanos al lenguaje natural.

<table>
<tr><td rowspan="8">Fomentar la programación y desarrollar lógica</td><td rowspan="7">Mecanica de programación</td><td rowspan="3">códificar es demasiado dificil</td><td>demostrar acciones en la interface</td><td>Pigmalion
Programming by rehearsal
Mondrian</td></tr>
<tr><td>Demostrar condiciones y acciones</td><td>AgentSheets
Chemtrains
StageCast</td></tr>
<tr><td>Especificar Accciones</td><td>Pinball construction set
Alternate Reality Kit
Klik and play</td></tr>
<tr><td rowspan="4">Mejorar los lenguajes de programación</td><td>Hacer mas comprensible el lenguaje</td><td>COBOL
Logo
Alice 98
HANDS</td></tr>
<tr><td>Mejorar la interacción con el lenguaje</td><td>Body Electric
Fabrik
Tangible programming with trains
Squeak eToys
Alice 99
Auto HAN
Physical Programming
Flogo</td></tr>
<tr><td>Integración con el entorno</td><td>Boxer
Hypercard
cT
Chart N Art</td></tr>
<tr><td rowspan="2">Actividades mejoradas por medio de la programación</td><td>Entretenimiento</td><td>Bongo
Mindrover</td></tr>
<tr><td>Educación</td><td>SOLO
Gravitas
StarLogo
Hank</td></tr>
</table>

Figura 3. Sistemas para fomentar la programación y desarrollar la lógica.
Adaptado de Kelleher y Pausch (2005)

Aún siendo extensiva, la taxonomía de Kelleher y Pausch (2005) puede complementarse con otros tipos de sistemas. Así, vemos que también hay aplicaciones que tienen como objetivo servir de complemento a cursos formales de programación, ya sea proporcionando ayudas visuales, proveyendo un andamiaje para colocar las estructuras de conocimiento conceptual o generando retroalimentación dirigida al simular la interacción de un tutor humano. Estos grupos adicionales se describen a continuación.

- Herramientas de visualización. La visualización de programas se define como la visualización del código en tiempo real de un programa o estructura de datos, mediante elementos gráficos. Urquiza-Fuentes y Velázquez-Iturbide (2009) probaron la usabilidad y efectividad educativa de 13 programas de visualización y propusieron una serie de características deseables en ellos. Algunos ejemplos de estas aplicaciones se pueden encontrar en Hamer, (2004) (visualización de de estructuras de datos en Java) y en Jehng, (1999) (enseñanza del tema de la recursividad). Pollack (2004) propuso una lista de criterios para ayudar en la selección de un sistema de visualización y así fomentar su uso.

- Objetos de aprendizaje. El objetivo básico de un objeto de aprendizaje consiste en que instructores puedan construir y distribuir pequeños componentes de software (relativos al tamaño de un curso) que puedan ser aplicados y reutilizados en distintos contextos instruccionales (Wiley 2000). Los objetos de aprendizaje tienen su origen en la teoría constructivista de aprendizaje (Gibbons 2000; Moisey 2003; Wiley 2000) y basan su estructura y diseño en los supuestos de dicha teoría. Sin embargo, en la literatura no se reporta suficiente evidencia empírica sobre su efectividad en el ámbito de la programación (Arevalo, Andrade, y Juan Gómez 2008).

- Mapas conceptuales. Los mapas conceptuales fueron desarrollados por Joseph Novak (Novak y Cañas 2006b; Novak 1998; Novak y Cañas 2006a). A su vez se basan en la teoría del aprendizaje significativo de David Ausubel (Ausubel 1963; Ausubel 1968) . Los mapas conceptuales son herramientas gráficas para organizar y representar jerárquicamente el conocimiento. Incluyen conceptos, y relaciones entre ellos, indicadas por una línea que los enlaza. Las palabras sobre la línea se conocen como “frases de enlace” y especifican la relación entre los dos conceptos asociados. Los conceptos representados (generalmente por medio de rectángulos o círculos) en el

mapa pueden ser enriquecidos con definiciones, imágenes y ejemplos que el estudiante puede consultar, al tiempo que obtiene una visión sinóptica del dominio de conocimiento consultado.

En el ámbito del aprendizaje de la programación, investigadores han reportado resultados favorables en el uso de mapas conceptuales, tanto como método de representación y evaluación del conocimiento conceptual (Keppens y Hay 2008) de alumnos, o como complemento para otras aplicaciones tales como los tutores inteligentes (Kumar 2006a; Maries y Kumar 2007).

- Tutores Cognitivos / Inteligentes. La literatura reporta un tipo especial de aplicación conocida como Tutor Cognitivo (J. R. Anderson, Corbet, y K. R. Koedinger 1995), que cuenta con un historial favorable en la enseñanza de temas como el álgebra y la geometría (K. Koedinger y J. Anderson 1997), la enseñanza de lenguajes de programación de inteligencia artificial, tales como LISP (Corbett 1993).

  Los principios detrás de este tipo aplicaciones se encuentran en arquitectura cognitiva conocida como ACT-R (Adaptive Control of Tought - Rational, J. Anderson, 2004; J.R. Anderson, 1996), que proporciona un marco de referencia elementar para comprender y simular los procesos cognitivos de la mente humana. Se parte del supuesto de que una *retroalimentación dirigida*, y una detección de los patrones de uso del aprendiz, mejorarán la transferencia de conocimiento. Los tutores cognitivos se encuentran entre la categoría de herramientas que cuenta más evidencia empírica de su efectividad (J. R. Anderson, Corbet, y K. R. Koedinger 1995; K. Koedinger y J. Anderson 1997; Kumar 2006b). Presentan, sin embargo la desventaja de ser costosos en su desarrollo. Aleven et al. (2006); Koedinger et al. (2004) han desarrollado a su vez herramientas para disminuir el esfuerzo y la complejidad de desarrollo de los tutores cognitivos.

Del universo de aplicaciones para apoyar el aprendizaje de la programación mostradas en este capítulo, son los tutores cognitivos los que reportan una mayor efectividad pedagógica. Su diseño orientado a la retroalimentación dirigida, bajo arquitecturas basadas en robustos modelos cognitivos ha producido resultados favorables. Sin embargo, además de que no se reportan tutores cognitivos diseñados para la enseñanza de la programación procedimental, y de que su costo de desarrollo es considerable, sus principios no están basados en la enseñanza explícita del conocimiento estratégico y metacognitivo, registrado tal como se genera por un programador experto, en su lenguaje natural.

La gran mayoría de las herramientas mostradas en esta sección tienen como objetivo crear modelos mentales en los estudiantes acerca de constructos básicos y/o lógica de programación. En esta tesis se argumenta que un elemento faltante en las herramientas existentes lo constituye la visualización explícita del conocimiento estratégico (según lo descrito en la sección 1.3.3) y metacognitivo (de acuerdo a lo desarrollado en la sección 1.3.4.) que están presentes en un programador con experiencia, lo cual representa un área de oportunidad de investigación y desarrollo para nuevos tipos de aplicaciones de software para apoyar el aprendizaje de la programación básica.

1.4 Justificación.

El aprendizaje de la programación es un tema cotidiano, complejo y relevante. Cotidiano, porque las materias introductorias de programación se imparten todos los días, tanto local como internacionalmente, en aulas de instituciones de nivel medio y superior. Anecdóticamente, cualquier instructor de programación puede relatar las dificultades que tanto él como sus alumnos experimentan en este proceso de enseñanza y aprendizaje.

El aprendizaje de la programación es también un tema complejo y de difícil solución, ya que en él intervienen –como se ha comentado– los procesos mentales del aprendiz (Fixx, Wiedenbeck, y Scholtz 1993; Ma et al. 2007), las estrategias pedagógicas seleccionadas por los instructores (Chieu 2007; Kumar 2006a; Rößling 2002), el entorno sociocultural que rodea al aprendiz (Booth 2001; Bruce 2004) y las aplicaciones de software implicadas, tales como lenguajes de programación y herramientas de apoyo al aprendizaje (Chansilp 2004; McKeown 2004).

En el ámbito socioeconómico, la programación es un tema relevante por el contexto de creciente demanda de empleos relacionados con el desarrollo de tecnologías de información y la decreciente oferta de empleos en los países llamados de primer mundo (Litecky, Prabhakatar, y Arnett 2006; de Raadt, Watson, y Toleman 2003; Koong, Lai C. Liu, y Xia Liu 2002).

Tal vez el aspecto en el cual el problema se manifiesta de manera más tangible, es en los altos porcentajes de reprobación que se reportan en las materias introductorias de programación a nivel universitario.

Así, en la literatura se puede leer que en años recientes hay índices de reprobación en estudiantes universitarios de primer año que cursan materias de programación cercanos al 30% (T. Boyle 2003; Guzdial 2002; UAA 2007). Otras fuentes aventuran cifras más alarmantes: entre un 30% y un 60% de toda la matricula de carreras de ciencias computacionales en cada universidad, falla en el primer curso de programación (Dehnadi y Bornat 2006). En síntesis, las facetas del problema que se han discutido en este capítulo y en las cuales se argumenta que la presente tesis puede aportar nuevo conocimiento para su posible solución, se muestran en la Tabla 1

Tabla 1. Justificación de la investigación.

Tema	Descripción	Posible aportación
Apoyo en la enseñanza tradicional	No suele enfatizarse lo suficiente el reforzamiento de modelos mentales válidos, la visualización explícita de estrategias de solución y el conocimiento metacognitivo.	La identificación de nuevos métodos didácticos que enfaticen tales elementos cognitivos faltantes.
Tasa de reprobación	Porcentajes entre el 30% y el 60% de acuerdo con las fuentes más pesimistas (Dehnadi y Bornat 2006)	Ayudar a disminuir los porcentajes de reprobación de alumnos de cursos introductorios universitarios a través del uso de artefactos de software.
Disponibilidad de herramientas	Aunque el catalogo de herramientas para apoyar el aprendizaje de la programación es amplio, las categorías de tales herramientas se han enfocado principalmente a facilitar la mecánica de la programación, a disminuir la complejidad de los lenguajes y a reforzar modelos mentales mediante ayudas audiovisuales e interactividad.	Se puede proponer una nueva clase de herramientas que permitan la visualización de estrategias de solución (los llamados planes o esquemas) y de elementos metacognitivos de depuración de errores y monitoreo del propio desempeño.

(…Continuación) Tabla 1. Justificación de la investigación.

Contexto socioeconómico	Existe poca oferta de programadores y una alta demanda de profesionales capacitados en el desarrollo de sistemas de información	Incrementar la oferta de programadores, fomentando la capacidad y el gusto por la programación en estudiantes novatos.

En esta tesis se parte del principio de que el diseño, desarrollo y uso de herramientas de software puede ayudar a mejorar el rendimiento en el aprendizaje de un alumno. Por lo que se argumenta puede ser de interés para la comunidad científica el proponer una nueva clase de artefacto de software cuyas propiedades estén relacionadas con los hallazgos de la literatura, considerados como críticos. En función de esto, se plantean los objetivos de la presente tesis en la siguiente sección.

1.5 Objetivos.

Con base en la premisa adoptada (herramientas de software pueden ayudar a mejorar el aprendizaje de las personas), el alcance de la presente tesis se circunscribe al planteamiento de una clase de artefacto de software cuyos atributos, proceso de construcción y prueba, puedan aportar conocimiento y elementos de solución al problema de investigación.

De tal manera, se optó por aplicar el marco conceptual y metodológico ISDT (Information Systems Design Theory o Teoría de Diseño de Sistemas de Información) como referencia para plantear las propiedades de tal clase de artefacto de software.

El antecedente del marco ISDT, puede encontrarse en el trabajo de Herbert A. Simon (Simon 1996), quien clasifica los tipos de investigación dentro de:

- Las ciencias naturales, que estudian los fenómenos que ocurren en el mundo (es decir, la naturaleza o la sociedad).
- Las ciencias del diseño o ciencias de lo artificial, que a su vez se caracterizan por:
    - Todo o parte del fenómeno puede ser creado artificialmente.
    - Estudiar objetos artificiales diseñados para alcanzar ciertos objetivos.

- Las ciencias sociales, que estudian procesos estructurales de un sistema social y su impacto en los procesos la organización social.
- Las ciencias del comportamiento, que estudian los procesos de decisión y las estrategias de comunicación dentro y entre los organismos en un sistema social.

En la literatura puede encontrase que el marco ISDT ha sido aplicado con éxito por investigadores en el ámbito de los sistemas de información para plantear nuevas clases de artefactos de software. Por ejemplo Markus, Majchrzak, y Gasser (2002), propusieron una Teoría de Diseño para “Sistemas que soportan procesos de generación de conocimiento emergente (EKP)”. Kasper (1996) propuso una ISDT para describir las propiedades de un sistema de soporte a la decisión (DSS) que alcanzara el objetivo de calibración perfecta de usuarios (una condición en la que la confianza del usuario acerca de una decisión sugerida por el sistema sería igual a la calidad real de tal decisión). Stein y Zwass (1995) desarrollaron una ISDT para Sistemas de Información de Memoria Organizacional (OMIS), que definieron como un tipo de sistema que busca extraer conocimiento del pasado para que se ajuste a condiciones del presente, para a su vez incrementar los niveles de efectividad de la organización.

En este contexto de teorías de diseño y habiendo señalado el problema de investigación e identificado los hallazgos relevantes de la literatura descritos en las secciones anteriores, los objetivos de la presente tesis son los siguientes:

Objetivo General.

- Aplicar la metodología ISDT para plantear una teoría de diseño de sistemas de información que proponga una clase de artefacto cuyas propiedades permitan la visualización de modelos mentales válidos, la visualización de estrategias de solución y el razonamiento metacognitivo de expertos al resolver problemas de programación, con la finalidad de fomentar un aprendizaje significativo y una transferencia de estrategias de solución de problemas en alumnos universitarios que estudian cursos introductorios de programación.

Objetivos Específicos.

1. Plantear una teoría de diseño de sistemas de información mediante la metodología ISDT, que proponga una clase de artefactos de software para fomentar un aprendizaje significativo en la aplicación de estrategias de programación para la escritura de programas básicos.
2. Diseñar y construir un artefacto de software derivado de la teoría de diseño planteada, que a su vez contenga propiedades sugeridas por otras teorías, tanto para el producto como para el proceso de diseño.
3. Probar empíricamente el artefacto construido bajo los principios de la teoría de diseño propuesta, tanto en sus propiedades de diseño, como en su efectividad para fomentar el aprendizaje.

1.6 Hipótesis.

El planteamiento de hipótesis es crítico en cualquier proceso de investigación. Su existencia obedece a una de las premisas básicas de la ciencia: la comprobación empírica derivada de la experimentación u observación. Después de la identificación del problema de investigación, las hipótesis –junto con los objetivos– definen el alcance y establecen los límites de cualquier estudio.

Pero en la ciencia existen variantes metodológicas. Por ejemplo, en la investigación cualitativa (vinculada a estudios dentro de las ciencias sociales), las hipótesis son sustituidas por proposiciones. En la investigación cuantitativa, se establecen hipótesis cuando el conocimiento existente permite formular predicciones acerca de la relación de dos o más variables. En las teorías de diseño de sistemas de información, la formulación de hipótesis está por supuesto presente, pero tiene una naturaleza dual: existen hipótesis de producto y de proceso y su formulación se realiza después de haber establecido los objetivos que debe alcanzar la clase de artefacto propuesta (llamados Meta-requerimientos) y las características con las que ésta debe contar (llamadas Meta-diseño).

El capítulo dos de esta tesis contiene una descripción detallada tanto de los antecedentes epistemológicos de las teorías de diseño de sistemas de información, como de sus componentes y etapas desde el punto de vista metodológico.

Dicho lo anterior, en la presente tesis es necesario aún definir una hipótesis general como elemento de comprobación tanto para los objetivos definidos, como para delimitar el problema general de investigación.

La hipótesis general del estudio es por tanto la siguiente:

Hipótesis: “Un artefacto de software que se derive de una teoría de diseño de sistemas de información (ISDT) para clases de artefactos diseñados para fomentar el aprendizaje significativo de estrategias de solución de problemas de programación, *fomentará un desempeño en el aprendizaje significativamente mejor* en alumnos universitarios que cursen materias introductorias de programación, en contraste con aquellos alumnos de las mismas características que no lo utilicen”.

Para que la anterior hipótesis pueda ser corroborada, la siguiente proposición deberá a su vez, ser probada como cierta:

Proposición uno: “El desempeño en el aprendizaje de un alumno puede medirse empíricamente en términos de su capacidad para escribir programas sintáctica y semánticamente correctos que solucionen problemas de programación básica”.

2 TEORÍAS DE DISEÑO DE SISTEMAS DE INFORMACIÓN (ISDT).

La creatividad consiste en permitirse cometer errores. El diseño consiste en saber cuáles conservar.

-Freeman Thomas

En 1992 Walls, Widmeyer, y El Sawy (Walls, et. al. 1992) publicaron el artículo seminal titulado "Building An Information System Design Theory for Vigilant EIS" (construyendo una Teoría de Diseño de Sistemas de Información para Sistemas Expertos Vigilantes), en donde plantearon los fundamentos del marco metodológico conocido como "Teorías de Diseño de Sistemas de Información" (Information Systems Design Theory o ISDT por sus siglas en inglés). Ellos definieron que las Teorías de Diseño de Sistemas de información son "Teorías prescriptivas que integran teorías normativas y descriptivas hacia rutas de diseño que tienen la intención de producir Sistemas de Información más efectivos" (Walls, Widmeyer, y El Sawy 1992, p.37).

Walls et. al. (1992) enfatizan que las teorías de diseño son prescriptivas (es decir, indican *que* hacer para alcanzar un objetivo), procedurales (señalan procedimientos), y combinadas (incluyen teorías de diversas ramas de la ciencia), y señalan que nunca deben involucrar predicción o explicación puras, teniendo que incluir pruebas experimentales para su validez[6].

El concepto de *artefacto* es descrito como algo que es artificial, o construido por humanos, en oposición a algo que ocurre naturalmente (Simon 1996). Así, una ISDT tiene que ver con el diseño, construcción y uso de artefactos basados en Tecnologías de Información (específicamente, software); aunque la naturaleza y

[6] Para una mayor explicación de la naturaleza de las teorías de diseño, ver ANEXO I.

alcance de lo que es un artefacto ha sido materia de cierto debate (Dahlbom 1996; Orlikowsky y Iacono 2001; Benbasat y Zmud 2003).

Investigadores como Hevner, March, y Park (2004) han extendido y refinado la idea original de Walls et. al., proveyendo una descripción más detallada del papel de la investigación de diseño[7] en el contexto de los sistemas de información, además de señalar una serie de principios y guías para evaluar y ejecutar este tipo de investigación. Por ejemplo, comentan que los artefactos generados por una ISDT pueden ser *constructos, modelos, métodos e instanciaciones*. Éstas últimas muestran "como los constructos, modelos o métodos pueden ser implementados *en un sistema ejecutable*. Demuestran factibilidad, permitiendo una evaluación concreta de lo apropiado que es un artefacto para su propósito definido, permiten a los investigadores aprender acerca del mundo real, ver como es afectado por el artefacto y como los usuarios responden a él". Hevner, et. al. (2004, p.79).

2.1 Componentes de una ISDT.

Walls. et. al. comentan que la palabra diseño, es tanto un verbo como un sustantivo, sugiriendo una naturaleza dual de objeto y proceso. Así, una Teoría de Diseño de Sistemas de Información tendría dos componentes básicos: *producto y proceso*. El primer aspecto se enfoca hacia las características de una clase de artefacto de software (producto de diseño) para un tipo especial de problema y el segundo hacia un proceso de diseño sugerido para construir tal clase de artefacto (proceso de diseño). Estos dos componentes constan a su vez de los siguientes elementos:

[7] El término original es "Design Research"

El producto del diseño.

- Meta-requerimientos. Los Meta-requerimientos describen una clase de objetivos hacia los que aplica la teoría. Se usa este término en lugar del concepto de "requerimiento", dado que se dice que una teoría de diseño no aplica a un solo problema o artefacto, sino para una clase de problemas.

- Meta-diseño. El meta-diseño describe la serie de características de una clase de artefacto de software que se hipotetiza va a satisfacer los meta-requerimientos. Nuevamente, se habla de meta-diseño, porque la teoría de diseño no se enfocará a un artefacto específico (por ejemplo, el sistema de control de inventarios o de nómina de una empresa X), sino a una clase de artefactos.

- Teorías núcleo. El tercer componente son las teorías núcleo, que se toman de las ciencias naturales y sociales y se supone gobiernan los requerimientos de diseño.

- Hipótesis comprobables del producto de diseño. El componente final, son hipótesis comprobables que se usan para verificar si el Meta-diseño satisface los Meta-requerimientos.

El proceso de diseño.

El segundo componente de una teoría de diseño de sistemas de información cubre el aspecto procedimental del diseño. Describe el método(s) sugerido(s) para construir la clase de artefacto propuesta.

- El método de diseño. Describe los procedimientos sugeridos que se hipotetiza son más eficaces para la construcción de la clase de artefacto de software.

- Teorías núcleo. Son teorías métodos que gobiernan el propio proceso de diseño. Se especifica que estas teorías núcleo pueden ser diferentes de aquellas asociadas al producto de diseño.

- Conjunto de hipótesis comprobables del proceso de diseño. Estas se usan para verificar si el método de diseño deriva en un artefacto que sea consistente con el Meta-diseño.

La síntesis de los componentes de una teoría de diseño de sistemas de información se muestra en la Tabla 2.

Tabla 2. Componentes de una Teoría de Diseño de Sistemas de Información (tomada de Walls et. al. 1992)

Diseño del producto	
1. Meta-requerimientos.	Describe la clase de objetivos para los que aplica la teoría.
2. Meta-diseño.	Describe la clase de artefactos que se hipotetiza alcanzarán los meta-requerimientos.
3. Teorías Núcleo.	Teorías de las ciencias naturales o sociales que gobiernan los requerimientos de diseño.
4. Hipótesis comprobables del diseño del producto.	Se usan para probar si el meta diseño satisface los meta requerimientos.
Diseño del proceso	
1. Método de Diseño	Una descripción de procedimientos para la construcción de artefactos
2. Teorías Núcleo	Teorías de las ciencias naturales o sociales que gobiernan el proceso
3. Hipótesis comprobables del diseño de procesos	Usadas para verificar si el método de diseño resulta en un artefacto que sea consistente con el meta-diseño

Según Walls et. al. (1992), la relación entre los componentes de una teoría de diseño de sistemas de información se esquematiza de acuerdo a la Figura 4:

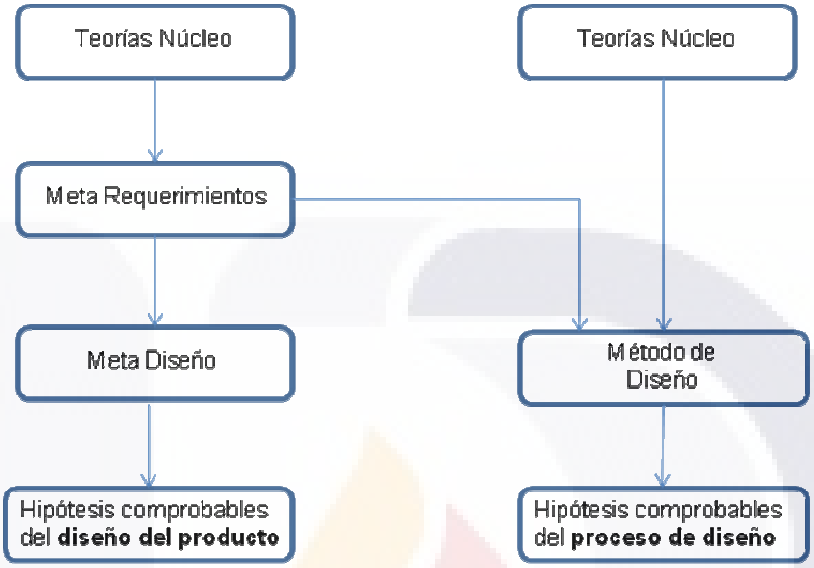


Figura 4. Relación entre componentes de una ISDT. Tomado de Walls, et. al. (1992)

2.2 Planteamiento de una ISDT.

Una dificultad inherente al desarrollo de Teorías de Diseño de Sistemas de Información, es la identificación de los Meta-requerimientos. Los Meta-requerimientos son el componente central de una ISDT y guían su planteamiento. Años después de la publicación de su artículo, al revisar el impacto de su propuesta original, Walls et. al. (1992), encontraron que "algunos académicos han tenido quejas acerca de lo incómodo que puede ser el enfoque ISDT y han señalado algunas de sus omisiones[8]" (Walls, Widmeyer, y El Sawy 2004, p.55).

[8] Por ejemplo, en su artículo Walls et. al., deducen Meta-requerimientos a partir de una serie *proposiciones*, que a la vez se extraen de la combinación de teorías núcleo relacionadas con un determinado problema de investigación. Lo que puede confundir es que los Meta-requerimientos se identifican a partir de teorías núcleo, pero no queda claro de donde surgen éstas últimas y porque las proposiciones no se muestran como un componente de una ISDT (ver Figura 5).

Indicadas estas salvedades, se deduce que el proceso general para plantear una ISDT consta de los siguientes pasos: (ver Figura 5).

1. **Descripción del problema**. En donde se identifican los temas y aspectos de la problemática de investigación sobre los que la literatura ha proporcionado mayor evidencia empírica.

2. **Identificación de teorías núcleo**. En donde se plantean proposiciones de teorías núcleo que sintetizan las facetas del problema y enumeran los elementos de las teorías núcleo que se argumenta pueden resolverlas. Un criterio discriminador de teorías núcleo consiste en identificar aquellas que reportan mayor evidencia de su validez y/o efectividad.

3. **Identificación de Meta-Requerimientos**. Derivado de las proposiciones de teorías núcleo planteadas en el paso anterior, se enumeran los Meta-requerimientos que el artefacto debe cubrir. Los Meta-requerimientos se redactan en términos prescriptivos (es decir, mandatorios).

4. **Planteamiento del Meta-Diseño**. En donde se enumeran las características funcionales con las que debe contar la clase de artefacto propuesta. Estas características de diseño se derivan a su vez de los Meta-requerimientos y deben acompañarse de una justificación también en un sentido funcional.

5. **Planteamiento de hipótesis comprobables de producto de diseño**. Se plantean hipótesis asociadas al producto de diseño, en términos de *factibilidad* de desarrollo y *efectividad* derivada de su uso. Se toman como referencia las características del artefacto enumeradas en el Meta-diseño.

6. **Identificación del Método de Diseño**. El enfoque del marco metodológico cambia del producto hacia el proceso. En este caso, se sugiere enfocar el método de diseño hacia teorías núcleo efectivas para la identificación de

requerimientos que se ajusten a las características del producto de diseño. Walls et. al. señalan que este componente se enfoca en la fase de determinación de requerimientos del ciclo de vida de los sistemas de información (Software Development Life Cicle, o SDLC por sus siglas en inglés).

7. **Planteamiento de hipótesis comprobables de proceso de diseño**. Finalmente, se plantean hipótesis para comprobar la *factibilidad* de aplicar el método de diseño propuesto para construir el artefacto (producto de diseño) descrito.

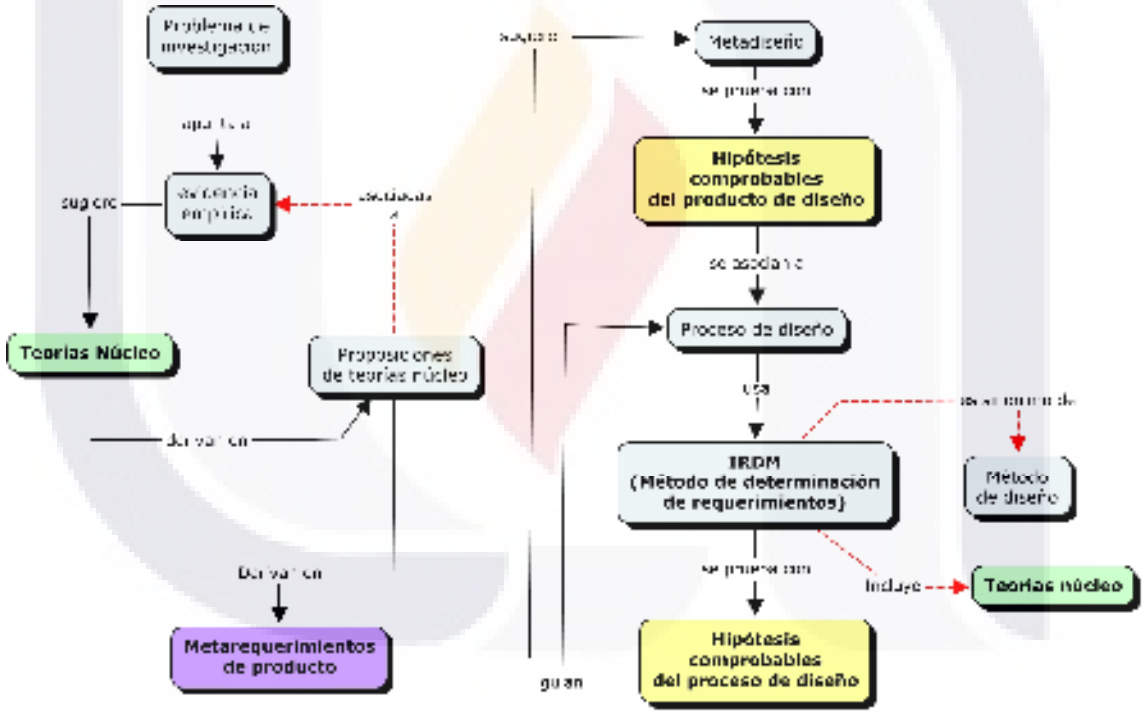


Figura 5. Planteamiento de una teoría de diseño de sistemas de información (ISDT)

Para instanciar su propuesta metodológica, Walls et. al. propusieron una ISDT para una clase de sistemas expertos que denominaron “VIS” (Vigilant Expert Systems, ver Figura 6). El punto de partida en este caso fue el problema de la efectividad en la toma de decisiones de ejecutivos acerca de asuntos estratégicos.

Para los fines de esta tesis, no se relatará aquí el detalle (dada su extensión) del proceso de instanciación descrito por Walls, et. al (1992). Baste comentar que cubre los siete pasos comentados en el proceso anterior, cada uno acompañado de una argumentación respaldada por hallazgos de la literatura.

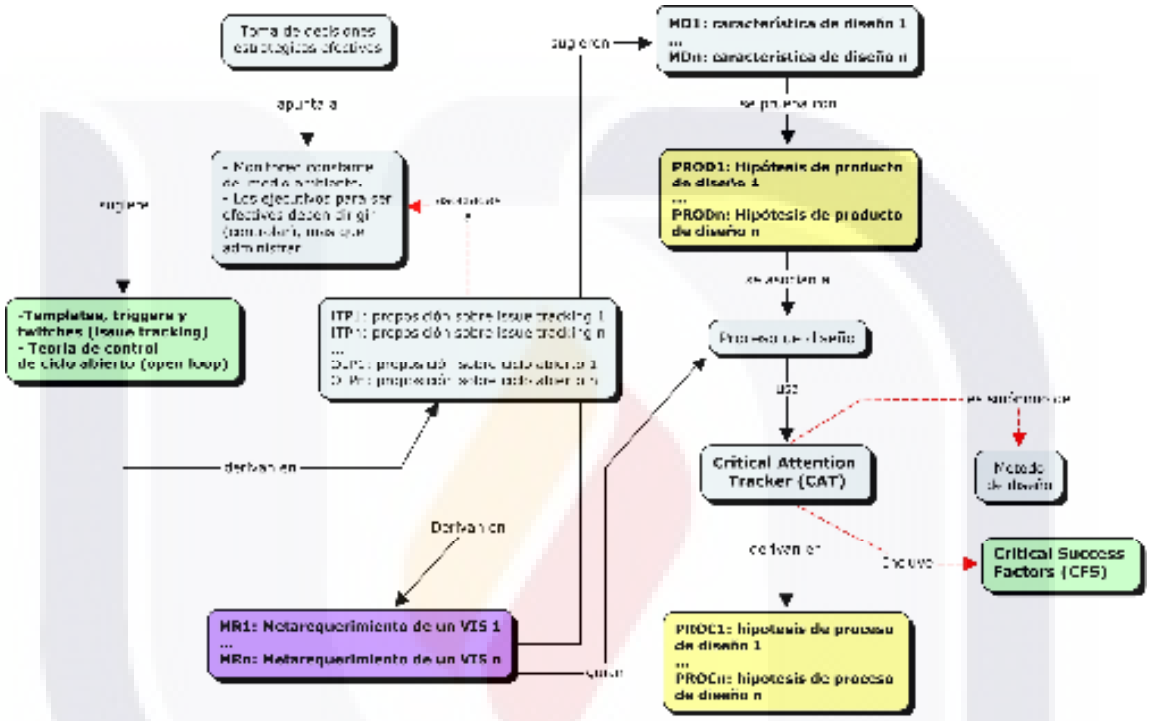


Figura 6. Planteamiento de una ISDT para un VIS. (Adaptado de Walls et. al., 1992)

3 TEORÍA DE DISEÑO DE SISTEMAS DE INFORMACIÓN PARA SISTEMAS VISORES DE PROTOCOLOS VERBALES.

3.1 Producto de diseño.

Creo en la evidencia. Creo en la observación, medición y razonamiento confirmados por observadores independientes. Creeré cualquier cosa, no importa lo salvaje y ridículo que sea, si existe evidencia de ello. Sin embargo, mientras más salvaje y ridículo sea algo, más sólida debe ser su evidencia.

-Isaac Asimov.

La evidencia derivada de la revisión de la literatura sobre el problema del aprendizaje de la programación apunta a dos temas recurrentes que hacen la diferencia entre programadores expertos y novatos: a) la posesión de modelos mentales válidos y b) la aplicación de estrategias efectivas de solución de problemas de programación (ver Figura 7). Estos temas se tomarán como punto de partida para el planteamiento de una ISDT para el diseño de una clase especial de artefacto de software que nombraremos Sistema Visor de Protocolos Verbales (SVP), por una de las teorías núcleo utilizadas.

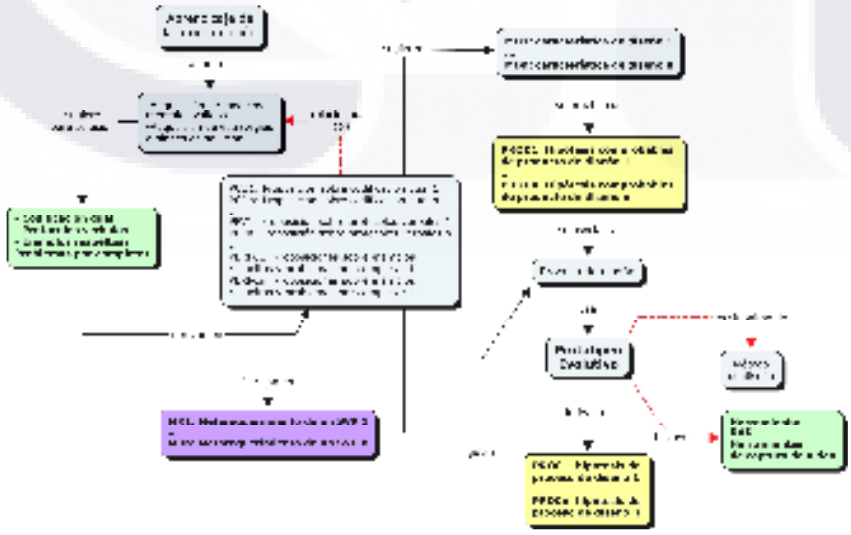


Figura 7. Planteamiento de una ISDT para un SVP

Una vez identificados los puntos críticos del problema de investigación, el punto focal para el planteamiento de una ISDT es encontrar teorías núcleo que:

a) Muestren evidencia de facilitar la adquisición de modelos mentales.
b) Indiquen métodos efectivos para registrar el proceso de pensamiento estratégico y metacognitivo de un sujeto identificado como experto en el dominio de la programación.
c) Faciliten el aprendizaje y transferencia de tales estrategias.

3.1.1 Teorías núcleo.

La mente humana tiene tres grandes sistemas de memoria que trabajan de manera combinada (ver Figura 8). En estos tres sistemas, la memoria de corto plazo o de trabajo juega el papel de intermediario entre la percepción del mundo exterior y los conocimientos, información y habilidades que tenemos previamente almacenados. Cabe mencionar que la distinción de estos sistemas de memoria no es por sí misma una teoría núcleo, pero si provee un punto de referencia y un lenguaje común a teorías discutidas en las secciones subsecuentes.

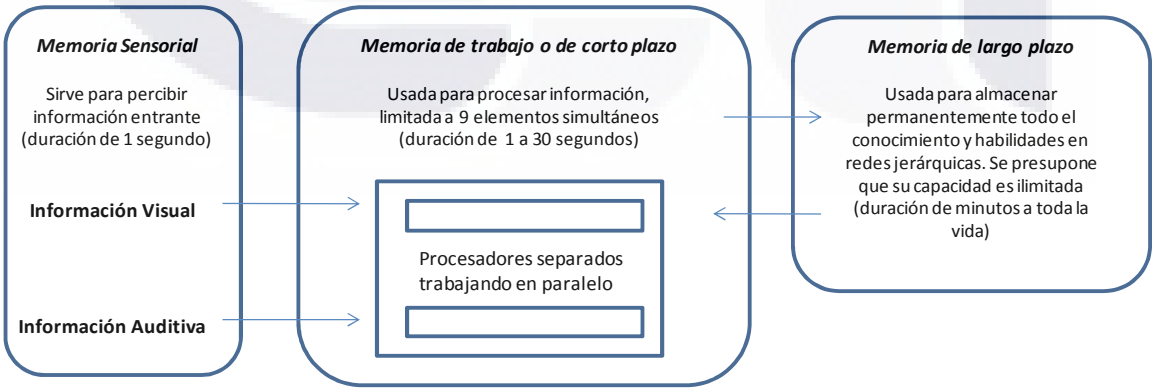


Figura 8. Los sistemas de memoria de la mente humana. Adaptado de Novak y Cañas (2006)

Así, la memoria de trabajo o de corto plazo, es un sistema que a su vez se divide en tres componentes. El sistema de ejecución central, que actúa como el sistema de control de atención, y dos sistemas esclavos: el área de trabajo Visio-espacial que manipula imágenes visuales, y el ciclo fonológico que almacena y ensaya la información basada en el lenguaje (ver Figura 9).

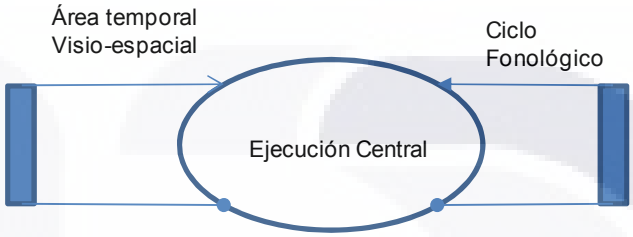


Figura 9. Modelo de la memoria de trabajo. Adaptado de Feinberg y Murphy (2000)

La memoria de largo plazo corresponde al inmenso cuerpo de conocimientos y habilidades que tenemos almacenado en nuestra mente, de formas relativamente accesibles. Todo lo que un ser humano sabe está guardado en la memoria de largo plazo, cuya capacidad parece ser ilimitada. El concepto de memoria de largo plazo comúnmente se asocia con el almacenamiento de conocimiento organizado en forma de estructuras esquemáticas y jerárquicas de conocimiento (Kalyuga 2007a).

La memoria sensorial, la cual registra los estímulos entrantes del medio ambiente por medio de nuestros sentidos, que incluyen la vista, el oído, el olfato, el gusto y el tacto. Las memorias sensoriales se extinguen rápidamente.

Es en la memoria de trabajo donde el aprendizaje tiene lugar. Pero una característica esencial de esta memoria es su limitada capacidad de almacenamiento, tanto en cantidad como en duración. Esto restringe el procesamiento de nueva información hacia el almacén masivo de información que es la memoria de largo plazo. La memoria de trabajo es entonces el componente activo en el cual la información nueva proveniente del medio ambiente se procesa

para formar estructuras de conocimiento que posteriormente se almacenarán en la memoria de largo plazo.

3.1.1.1 Teoría de Codificación Dual.

La primer teoría núcleo que se propone para la ISDT para desarrollar sistemas que apoyen el aprendizaje de la programación es la llamada Teoría de Codificación Dual (Paivio 1990; Sadoski y Paivio 2004) ó Dual Coding Theory (DCT) por sus siglas en inglés. Esta teoría propone que el sistema de memoria de corto plazo humano consta a su vez de dos grandes subsistemas: uno verbal y uno visual. El subsistema visual procesa y almacena información concreta, tal como imágenes y sonidos. El subsistema verbal se encarga de procesar el lenguaje y la información abstracta. De acuerdo con esta teoría, ambos sistemas son independientes, pero conectados entre sí. A la creación ya sea de una representación verbal a partir de un estímulo visual, o de una imagen a partir de lenguaje, se le conoce como *conexión referencial* (ver Figura 10).

En la DCT, cuando la información es registrada tanto de manera visual como verbal se dice que está “codificada dualmente” (Kuo y Hooper 2004; Meyer y Sims 1994; Sadoski y Paivio 2004). Uno de los principales postulados de la teoría indica que la información se recuerda mejor si es almacenada dualmente, porque al perderse una parte (ya sea verbal o visual) la otra se encuentra disponible. Relacionado con este principio, se dice también que las imágenes se recuerdan más fácilmente que las palabras porque es más probable que éstas activen las conexiones referenciales.

Como consecuencia de lo anterior, se dice que es más probable que ocurra la codificación dual cuando un material instruccional es susceptible de representarse de manera concurrente con imágenes y texto (Paivio, 1986;

Sadoski, Goetz, & Avila, 1995). De acuerdo con la teoría, el aprendizaje de conceptos concretos es más sencillo que el de conceptos abstractos, dado que los primeros son almacenados como imágenes y los segundos como representaciones verbales, los cuales tienen menos acceso a las codificaciones no verbales.

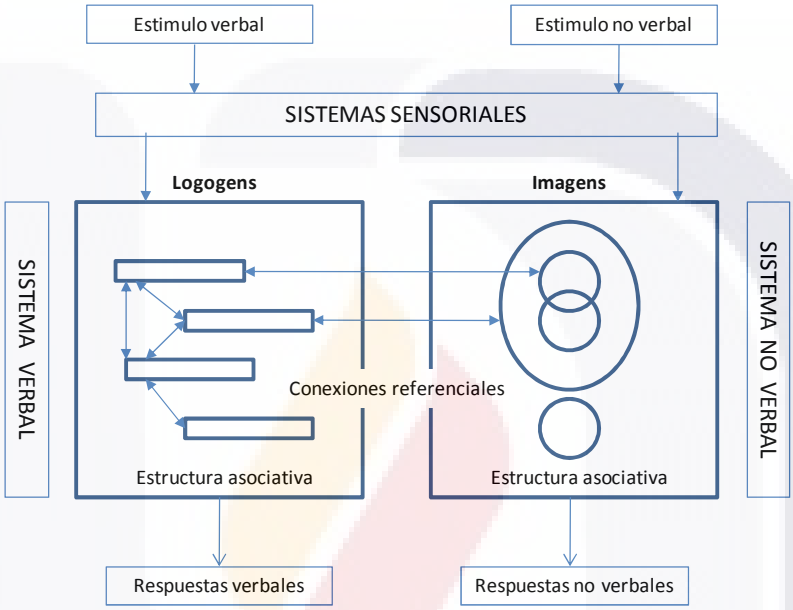


Figura 10. Teoría de la codificación dual. Adaptado de Sadoski y Paivio (2004)

En su teoría, Paivio indica que los dos códigos mentales (verbal y visual) y nuestros cinco sentidos son ortogonales (ver Tabla 3). Esto significa que los dos códigos tienen subconjuntos de representaciones mentales que son cualitativamente diferentes, dadas las experiencias sensoriales de las cuales se originaron. Es decir, el ser humano desarrolla representaciones visuales en el código verbal para unidades de lenguaje que ya hemos visto, tales como letras, palabras o frases (el ejemplo dado por Paivio es un bate de beisbol) y también se desarrollan representaciones visuales en el código no verbal, para formas no lingüísticas que hemos visto tales como objetos comunes o escenas (p.ej. un bate de madera o uno de aluminio).

Similarmente, se desarrollan representaciones auditivas en el código verbal para unidades de lenguaje que hemos escuchado, tales como fonemas y sus combinaciones (siguiendo a Paivio, el fonema /b/, la silaba /-ba/, la palabra /bate/) y representaciones auditivas en el código no lingüístico acerca de sonidos ambientales que hemos escuchado (el crack de un bate de madera golpeando una pelota). También se desarrollan representaciones táctiles (llamadas hápticas) o cinéticas en el código no verbal para actos motrices (como pronunciar /b/ o escribir la letra b o tocar la letra braile /b/), y desarrollamos representaciones hápticas para la "sensación activa" de objetos, texturas y movimientos (p.ej. el swing al batear). No representamos lenguaje en las modalidades en el sentido químico (gusto, olfato), pero tenemos representaciones no verbales para ellas (el olor y sabor de una salchicha en un juego de beisbol).

Tabla 3. Relación ortogonal entre los códigos mentales y los cinco sentidos en la DCT (Tomada de **Sadoski y Paivio (2004)**

	Códigos mentales	
Modo sensorial	**Verbal**	**No verbal**
Visual	Lenguaje visual (Escritura)	Objetos visuales
Auditivo	Lenguaje auditivo (Habla)	Objetos ambientales
Táctil	Braille, escritura manual	Sensación de objetos
Gustatorio	---	Recuerdos sobre sabores
Olfatorio	---	Recuerdos sobre olores

En la DCT las unidades básicas del sistema verbal se conocen como *logogens* y las unidades básicas del sistema no verbal, se conocen como *imagens* (ver Figura 10).

De acuerdo con la DCT, un logogen es cualquier cosa aprendida como una unidad de lenguaje en alguna modalidad sensorial. Las unidades de lenguaje varían en tamaño, aunque algunos tamaños son más familiares que otros (por ejemplo las palabras). Por lo tanto, tenemos *logogens* visuales para palabras

escritas y frases; *logogens* auditivos para fonemas y pronunciaciones de frases; *logogens* hápticos para pronunciar, escribir o firmar estas unidades de lenguaje.

Los *imagens* son específicos a una modalidad y también varían en tamaño y tienden a ser percibidas en conjuntos anidados. Es decir, a veces las imágenes mentales frecuentemente están incluidas unas dentro de otras.

Ambos tipos de representaciones pueden activarse de distintas maneras. Los logogens pueden activarse por medio de entradas sensoriales directas (tales como el lenguaje impreso) y los imagens pueden activarse al ver objetos familiares. Sin embargo, ambos tipos de representación mental pueden activarse indirectamente, como cuando *espontáneamente* convertimos imágenes en palabras o cuando nombramos objetos.

Evidencia sobre la Teoría de Codificación Dual.

Datos empíricos parecen apoyar los postulados de la DCT. Por ejemplo Meyer y Sims (1994) diseñaron experimentos en un estudio efectuado con estudiantes universitarios sin conocimiento previo de dispositivos mecánicos con diversos tipos de material instruccional (texto, imágenes, texto e imágenes mostrado de manera concurrente y texto e imágenes mostrados de manera secuencial), encontrando que sus resultados soportan la teoría de codificación dual, que enfatiza la construcción de conexiones mentales entre representaciones visuales y verbales. En otro estudio Kuo y Hooper (2004) estudiaron el efecto de material instruccional en el aprendizaje de caracteres chinos. Crearon cuatro grupos experimentales y los participantes que utilizaron material codificado dualmente se desempeñaron mejor que los demás.

Kounios y Holcomb (1994), utilizando ERPs[9] encontraron evidencia que soporta la Teoría de Codificación Dual en experimentos con sujetos realizando actividades de clasificación concreta-abstracta y decisiones léxicas. En aún otro estudio, Mayer y Richard B. Anderson (1991), realizaron dos experimentos con estudiantes sin conocimiento de mecánica, mostrando animaciones que indicaban el funcionamiento de una bomba de bicicleta con descripciones verbales mostradas antes (palabras antes de las imágenes) y durante la animación (palabras junto con las imágenes). El grupo usando palabras junto con las imágenes resultó más eficiente que el primero.

3.1.1.2 Protocolos verbales.

Relacionado con el problema de representar con fidelidad el conocimiento estratégico de un programador experto, la siguiente teoría núcleo propuesta es el método de los protocolos verbales. El pensamiento en voz alta como método de representación y análisis de procesos de pensamiento tiene sólidas bases científicas en el ámbito de la psicología cognitiva (Ericsson y Simon 1993; Newell y Simon 1972; Russo, Jonson, y Stephens 1989). Inicialmente fue desarrollado para el estudio de los procesos de la memoria de corto plazo de las personas (a que cosas prestan atención y en qué orden) cuando éstas llevan a cabo tareas de resolución de problemas.

Cabe hacer notar que el análisis de protocolos verbales es una técnica muy usada en los estudios sobre el aprendizaje de la programación, en donde suelen recolectarse muestras de protocolos verbales de grupos de personas (ya sea novatos, expertos o ambos) al momento de diseñar programas, para luego

[9] Event Related Potentials o Potenciales Relacionados a Eventos. La medición de ERPs es obtenida usando electroencefalogramas para monitorear la actividad cerebral asociada a ciertos estímulos.

verificar la existencia de patrones de diseño (Détienne 1995; Burkhardt, Détienne, y Wiedenbeck 1997; Rist 1996; Rist 1989); inclusive, los protocolos verbales también están siendo explorados como técnica didáctica en algunos cursos introductorios de programación, con resultados favorables (Arshad 2009). La verbalización también se usa con frecuencia en estudios relacionados con la evaluación de la usabilidad (J. Nielsen 1993; Krahmer y Ummelen 2004; Knox, Bailey, y Lynch 1989), en el contexto de la ingeniería de software.

En un sentido amplio, los protocolos verbales documentan, (mediante mediante grabaciones de audio y/o video que luego son transcritos), el comportamiento mental de una persona al solicitársele que "hable en voz alta" mientras lleva a cabo una tarea en particular. Posteriormente estos documentos transcritos se analizan para estudiar el proceso de pensamiento seguido por la persona.

Ericsson y Simon (1980) plantearon que los datos verbales son una fuente válida de comprensión de los procesos cognitivos humanos tanto para verificar (dentro de la interpretación de un marco teórico) como para descubrir fenómenos de interés. Ericsson y Simon argumentan que una oración es la representación de una idea y que puede usarse para identificar distintos tipos de información y distintos tipos de procesos cognitivos. También distinguen –por un lado– entre la introspección clásica, los reportes retrospectivos y la comunicación con el experimentador, y por el otro la verbalización de pensamientos que reflejan la atención hacia una acción que se está realizando.

Nielsen, Clemmensen, y Carsten (2002) comentan que esta diferenciación tiene que ver con la memoria de corto plazo, bajo el supuesto de Ericsson y Simon de que todo lo que sabemos ha pasado por nuestra memoria de corto plazo y hemos sido consientes de ello. Podemos verbalizar lo que percibimos mientras lo estamos percibiendo y podemos verbalizar aquello sobre lo que fuimos consientes si se nos cuestiona sobre ello corto tiempo después de que sucedió, dado que lo

tenemos aún en nuestra memoria de trabajo. Sin embargo, si ha pasado un lapso de tiempo entre la percepción y la petición de recuerdo, produciremos descripciones y explicaciones, no un reporte de nuestros pensamientos inmediatos, porque la información en la memoria de corto plazo se ha perdido. La consecuencia de este supuesto fundamental –y debido a que el objetivo es estudiar los procesos cognitivos dirigidos por tareas–, es que solamente ciertos tipos de verbalización retroactiva proporcionarán la información utilizada mientras se realiza una tarea dada. El interés de los autores (y para el caso, de cualquier investigador) es identificar y analizar estas verbalizaciones.

Ericsson y Simon distinguen tres tipos de verbalizaciones:

- Los 'Talk alouds' que son vocalizaciones de pensamientos que ya están codificados en forma verbal. Los "Talk alouds" recuperan procesos cognitivos que operan directamente sobre información codificada oralmente. La verbalización inicia inmediatamente y procede como una vocalización del habla interna. Los protocolos tomarán la forma de una entrega en serie de códigos orales, por ejemplo números, letras, etc.
- Los "Think-alouds", que constan de constan de secuencias de pensamientos que contienen decisiones y conceptos previamente almacenados. En un protocolo de forma "Think-aloud", se le pide al sujeto que realice cierta tarea y que verbalmente explique el método que está aplicando para resolverla. Los protocolos tomarán la forma de oraciones que pueden ser comprendidas como pensamientos dentro o fuera del contexto de otros pensamientos.
- Los Reportes retrospectivos, que tratan sobre pensamientos no almacenados en la memoria de corto plazo. Los reportes retrospectivos producen un resultado similar que los Think alouds, excepto que son más coherentes y más susceptibles a errores si se comparan con lo que el sujeto realmente observó e hizo durante la sesión.

Por otro lado, Ericcson y Simon señalan, que si un sujeto verbaliza una frase tal como "¿qué es lo que estoy pensando?" esto sugiere que dicho sujeto tiene acceso poco frecuente a pasos intermedios de ese proceso de pensamiento debido a un proceso de automatización o por algún tipo de pensamiento estratégico o de meta nivel, en lugar de "reportar información acerca de lo que se está poniendo atención" (Ericsson y Simon 1993, p.244). Por lo tanto, para los objetivos de nuestra teoría de diseño, si el objetivo es precisamente mostrar estos procesos de pensamiento estratégico, este es un punto al que se debe poner especial interés, sobre todo al momento de seleccionar sujetos suficientemente experimentados y que sean capaces de reconocer sus propios procesos metacognitivos, además de instruirlos previamente acerca de no omitir estas verbalizaciones al momento de grabar los protocolos.

Transcripción de un protocolo verbal

En un inicio, el objetivo del trabajo Ericsson y Simon fue el análisis formal de reportes verbales en el contexto de la psicología cognitiva y su método de análisis de protocolos se desarrolló con el propósito de poder modelar procesos cognitivos guiados por tareas. Sin embargo, su técnica ha sido retomada en diversos ámbitos, más allá de la psicología cognitiva. Como ya se ha dicho, los protocolos verbales son muy usados en el contexto de la ingeniería de software, específicamente en la rama de pruebas de usabilidad (Knox, Bailey, y Lynch 1989; Krahmer y Ummelen 2004; Wells 2006). Un breve resumen del proceso de elaboración de un protocolo verbal se indica a continuación (adaptado de Nielsen, et. al. (2002)).

a) Transcripción. Ericcson y Simon encontraron que el surgimiento de nuevas tecnologías (tales como la grabadora de audio a mediados del siglo XX) incrementaron la posibilidad de tratar los protocolos verbales como datos

brutos, ya sea en la forma de cintas de audio o como transcripciones verbales de tales cintas. Pero advierten que "los datos no hablan por sí solos" especialmente en un sistema de memoria en donde no se pueden tener observaciones de primera mano y que no pueden ser exactamente reproducidos. Ericcsson y Simon señalan que los datos siempre deben ser codificados e interpretados dentro del marco de una estructura teórica.[10]

b) Segmentación. Ericsson y Simon comentan que los protocolos deben ser segmentados de tal forma que cada verbalización constituya una instancia de un proceso general, sugiriendo que pistas apropiadas pueden ser "pausas, entonaciones o marcadores gramaticales para frases y oraciones completas" es decir, las pautas para segmentar un discurso ordinario.

c) Codificación. Un esquema de codificación debe desarrollarse a-priori y el vocabulario debe desarrollarse para a) un análisis de tareas inicial y b) realizar un examen preliminar de los protocolos. Sin embargo los autores notan que algunas tareas tienen un lenguaje preciso de comunicación y otros no comparten un vocabulario común. Este es un problema recurrente en algunos dominios cuando se trata de establecer lo que sucede dentro de las cabezas de las personas, por ejemplo, cuando se habla de cooperación en el lenguaje del ejercito, es difícil hablar de cooperación, dado que la milicia usa un lenguaje de comandos y órdenes, no uno de cooperación.

3.1.1.3 Efecto del ejemplo resuelto y el problema por completar.

El llamado efecto del ejemplo resuelto[11] es descrito por John Sweller en el contexto de su Teoría de Carga Cognitiva (J. Sweller, van Merrienboer, y Paas

[10] Esto resulta necesario en el contexto interpretativo de psicología cognitiva. En el ámbito de ingeniería de software, el punto focal suele ser la aplicación que se está probando y la terminología es aquella relacionada con el artefacto sujeto de estudio.

[11] El término en inglés es "Worked Example"

1998; John Sweller 1994). Los ejemplos resueltos enfocan la atención del estudiante hacia los diversos estados de un problema y sus pasos de solución, habilitando en los aprendices –por inducción– la adquisición de estrategias de solución no generales. De acuerdo con Sweller, este razonamiento lleva a una predicción anti-intuitiva: estudiar ejemplos resueltos puede facilitar la construcción de esquemas y la transferencia de habilidades, más que la propia solución de esos problemas. Esta predicción se explica en términos de una "reducción de la carga cognitiva" (J. Sweller 1988b), dado que estudiar un ejemplo resuelto impone menos esfuerzo mental al estudiante novato, que resolver el problema por sí mismo, usando estrategias superficiales de solución de tipo prueba y error.

La efectividad de los ejemplos resueltos como método instruccional está bien documentada en la literatura (Gerjets, Scheiter, y Catrambone 2004; Renkl, Hilbert, y Schworm 2009; Renkl, Atkinson, y Große 2004; M. Ward y J. Sweller 1990; J. Sweller y G. Cooper 1985). Parece existir un acuerdo general acerca de que los ejemplos resueltos son una estrategia efectiva de adquisición de habilidades de resolución de problemas para estudiantes principiantes, en cualquier dominio. Se argumenta que estudiar un ejemplo resuelto enfoca la atención del estudiante tanto hacia las características estructurales clave que definen a que categoría particular pertenece el problema, como hacia su estrategia de solución.

En el diseño de ejemplos resueltos, es deseable que se identifiquen las características críticas de los mismos, acompañándolo con anotaciones de aquellos aspectos clave que se supone están tratando de ilustrar (J. R. Anderson, C. Boyle, y Corbell 1990). Sweller propone que en el ejemplo resuelto la atención del estudiante sea llevada hacia los componentes que llevan a la construcción de estrategias de solución. Por ejemplo: a qué tipo de problema pertenece el ejemplo, que características estructurales tiene y que procedimientos de solución específicos se usaron de acuerdo a su categoría.

Evidencia sobre ejemplos resueltos

Sweller, van Merrienboer, y Paas (1998) proporcionan numerosas referencias acerca de eficacia de los ejemplos resueltos como método de enseñanza. Por ejemplo, Sweller y Cooper (G. Cooper y J. Sweller 1987; J. Sweller y G. Cooper 1985) documentaron su efectividad para el aprendizaje y resolución de nuevos problemas de algebra. Zhu y Simon (1987), en un estudio longitudinal, encontraron que los ejemplos resueltos podían ser un reemplazo para la enseñanza convencional, encontrando, entre otras cosas, que un curso de matemáticas podía ser completado en dos años, en lugar de tres, si se hacía énfasis en ejemplos resueltos. Trafton y Reiser (1993) encontraron que estudiantes universitarios al usar ejemplos resueltos acerca del lenguaje de programación de inteligencia artificial LISP, se desempeñaron mejor que aquellos que resolvían por si mismos ejemplos equivalentes.

En estudios más recientes, (Renkl, Hilbert, y Schworm 2009) encontraron que los ejemplos resueltos también son efectivos dominios de conocimiento heurístico (por ejemplo, comprobación de axiomas matemáticos, solución de problemas en aparatos), además de los dominios de conocimiento en los que tradicionalmente se aplican, de naturaleza algorítmica.

Limitaciones de los ejemplos resueltos

La aplicación de ejemplos resueltos como método de aprendizaje debe hacerse cuidadosamente. VanLehn (1996) muestra evidencia de que aprender de ejemplos resueltos es crítico en dominios bien estructurados (tales como las matemáticas, la física o la programación), pero que éstos deben usarse solamente durante las primeras etapas de adquisición de habilidades, cuando el aprendiz está iniciando su aprendizaje del dominio.

Relacionado con lo anterior, Anderson, Fincham, y Douglass (1997) presentaron un modelo de cuatro etapas de adquisición de habilidades en el cual los estudiantes que están en la etapa inicial, aprenden por analogía utilizando ejemplos de problemas resueltos y tratando de asociarlos con los nuevos problemas que se les piden resolver. En la etapa siguiente los estudiantes ya han adquirido algunas reglas abstractas de solución que los guían en la solución de problemas (llamadas reglas de conocimiento declarativo), en la tercera etapa, estas reglas se "proceduralizan" y en la cuarta, estas reglas se vuelven automáticas y el aprendiz ya cuenta con un amplio conjunto de reglas que puede utilizar y combinar para solucionar nuevos problemas. En este contexto, nuevamente, encontraron que es solamente en las primeras etapas de adquisición de habilidades cuando el uso de ejemplos resueltos es efectivo.

Esta distinción entre estudiantes en fases iniciales y fases posteriores en el proceso de adquisición de habilidades es crítico para la efectividad de los ejemplos resueltos, pudiendo incluso tener efectos de adversos si se aplican a estudiantes de alta habilidad. Investigadores han encontrado evidencia sobre un "efecto de retroceso de habilidades[12]" (Kalyuga 2007b; Kalyuga et al. 2001), que sucede cuando el material instruccional se presenta de manera redundante, en donde al mostrar ejemplos resueltos a estudiantes que ya conocen el proceso de solución, se impone un esfuerzo cognitivo adicional.

También, se advierte que un uso excesivo de ejemplos resueltos puede fomentar en los aprendices la transferencia de patrones de solución estereotipadas que inhiban la generación de soluciones nuevas y creativas (Smith, T.B. Ward, y Schumacher 1993).

Otra desventaja es que los ejemplos resueltos no obligan a los estudiantes a estudiarlos cuidadosamente (Sweller, et. al. 1998). Es decir que estudiantes con

[12] El término en inglés se conoce como "Expertise Reversal Effect"

mayor habilidad y motivación pueden procesar completamente, e incluso elaborar nuevos ejemplos resueltos, pero los estudiantes de menor voluntad de estudio y de perfil táctico, tienden a estudiarlos solo cuando tienen problemas para resolver los problemas convencionales (Chi et al. 1982; LeFevre y Dixon 1986). Sweller advierte que consultar ejemplos resueltos *al mismo tiempo* que se trata de resolver un problema convencional impone un mayor esfuerzo y carga cognitiva al estudiante, haciendo que la memoria de trabajo trabaje en paralelo, provocando una posible sobrecarga.

Problemas por completar

Como alternativa a los ejemplos resueltos, van Merrienboer y Krammer (1987, 1990) propusieron el uso de "problemas por completar[13]". Los problemas por completar son problemas en los que se cuenta con un estado de avance parcial y un estado objetivo deseado y se solicita a los aprendices que provean una o varias soluciones parciales o intermedias.

Los ejemplos por completar son efectivos en dominios orientados al diseño, tales como el diseño de software, el diseño de circuitos electrónicos, procesos de planeación de la producción, programación de control numérico y arquitectura (Sweller et. al., 1998). Más importante, los problemas por completar proveen un puente entre los ejemplos resueltos y los problemas convencionales. Es decir que los ejemplos resueltos son problemas con soluciones completas y los problemas convencionales son problemas por completar sin una solución provista.

Así, una estrategia instruccional que inicia con ejemplos resueltos, continua con problemas por completar que provean casi toda la solución y gradualmente

[13] El término en inglés es "Completion Problems"

avanza hacia problemas por completar en los que el estudiante provea casi toda la solución se conoce como "estrategia de completar[14]" (Sweller, et. al. 1998, p.247).

Evidencia sobre la efectividad de los problemas por completar

Van Merrienboer (1990) reportó el uso de problemas por completar en un curso de introducción a la programación de educación media, comparando los resultados de usar una estrategia usando problemas por completar y resolución de problemas convencionales. El rendimiento del grupo que usó problemas por completar fue superior en términos de construcción de nuevos programas. El estudio fue posteriormente replicado en un entorno de educación superior (van Merrienboer y De Croock 1992) con resultados similares.

Las estrategias basadas en problemas por completar continúan siendo aplicadas y estudiadas en años recientes tanto en el ámbito de la enseñanza de la programación (Caspersen y Bennedsen 2007) como en dominios similares a la programación como la enseñanza de la estadística y la física (Renkl, Atkinson, y Große 2004), como en dominios de naturaleza heurística como la comprobación de teoremas matemáticos (Renkl, Hilbert, y Schworm 2009).

3.1.2 Metarequerimientos

Continuando con el planteamiento de la ISDT para Sistemas Visualizadores de Protocolos, para la identificación de los Meta-requerimientos, es necesario hacer un estudio detallado del problema de investigación, específicamente sobre aquellos aspectos que cuentan con mayor evidencia empírica. Después se procede a enfocar la búsqueda hacia teorías núcleo que puedan aplicarse a la

[14] El término en inglés es "Completion Strategy"

solución de la problemática (como las descritas en las secciones anteriores), en donde estas teorías núcleo deben contar con razonable evidencia de su validez y/o efectividad.

Posteriormente, el planteamiento de las *proposiciones de teorías núcleo* es el punto intermedio entre el estudio del problema de investigación y la búsqueda de teorías núcleo (ver Figura 5). A su vez, las proposiciones son el paso anterior para poder definir los Meta-requerimientos. Así se obtiene una liga entre el problema de investigación, la teoría y el diseño. Entre problemática y ciencia.

En las secciones (3.1.1.1, 3.1.1.2 y 3.1.1.3), se describieron las teorías núcleo que se utilizarán para la ISDT de Sistemas Visualizadores de Protocolos (SVPs): La Teoría de Codificación Dual, Los Protocolos Verbales y una estrategia instruccional que combine Ejemplos Resueltos y Problemas por Completar. A continuación se plantean las proposiciones que derivarán en los Meta-requerimientos de la ISDT para SVPs.

Como ya se ha dicho, un modelo mental es la representación interna de una tarea o sistema complejo, cuya construcción permite a una persona predecir y comprender su funcionamiento (George 2000).

En programación, un modelo mental se refiere a la imagen que tiene el usuario sobre el procesamiento invisible que ocurre al interior de la computadora entre una entrada y una salida (Bayman 1983). Contar con modelos mentales válidos sobre estructuras de programación es crítico (Winslow 1996, p.21). Estos modelos mentales deben proveerse al estudiante mediante descripciones explícitas (Du Boulay 1989, pp.281-290).

Al iniciar el estudio de la programación, los programadores novatos pueden crear modelos mentales no válidos (Winslow 1996; Ben-Ari 1998; Dehnadi y Bornat 2006; Fixx, Wiedenbeck, y Scholtz 1993). Por lo tanto, derivadas de la

discusión anterior e integrando los principios de la Teoría de Codificación Dual (ver sección 3.1.1.1), se plantean las proposiciones de la Tabla 4.

Tabla 4. Proposiciones de Codificación Dual.

	Proposiciones sobre Codificación Dual
PCD1	Un modelo mental válido puede ser adquirido con mayor eficacia por un estudiante de programación si éste lo codifica dualmente.
PCD2	Los modelos mentales de un experto pueden mostrarse explícitamente mediante texto explicativo (código verbal) e imágenes del código (código visual).

Un componente cognitivo faltante en los programadores que inician el estudio de la programación, son las llamadas estrategias o planes (Fixx, Wiedenbeck, y Scholtz 1993). Existe evidencia empírica acerca de que la estrategia es el componente cognitivo básico usado para diseñar y comprender programas (Rist 1995, p.514).

Las estrategias sirven para manejar situaciones estereotípicas que ocurren durante la solución de un programa (Soloway y Ehrlich 1984; Brooks 1990). Son procesos exploratorios e incrementales, determinados por episodios menores de resolución de problemas y re-evaluaciones frecuentes del avance hacia el problema final. Los programadores expertos tienen estrategias de solución predefinidos para tipos de problemas de programación recurrentes. Generalmente estas estrategias no pueden ser deducidas de un programa terminado, que puede dar mucha información sobre conceptos de sintaxis, pero no sobre las estrategias que se usaron para escribirlo.

Por medio de la metacognición una persona identifica la naturaleza de un problema, selecciona una representación mental útil, selecciona la estrategia más efectiva para ejecutarla y presta atención a la retroalimentación sobre cómo se está avanzando en la resolución del problema (Gourgey 1998); Miles et al. (2003)) comentan que el conocimiento estratégico incluye habilidades meta-cognitivas

tales como la prueba de programas, la depuración de errores de lógica y el monitoreo de la efectividad de las estructuras sintácticas utilizadas.

Entonces, derivada de la discusión precedente e integrando los principios detrás de los Protocolos Verbales (ver sección 3.1.1.2), se plantean las proposiciones de la Tabla 5.

Tabla 5. Proposiciones sobre protocolos verbales.

Proposiciones sobre protocolos verbales	
PPV1	Las estrategias de solución de programadores expertos se pueden mostrar explícitamente a estudiantes de programación por medio de protocolos verbales.
PPV2	Los modelos mentales de un experto pueden mostrarse explícitamente a estudiantes de programación si los segmentos de un protocolo verbal se acompañan de imágenes o animaciones
PPV3	La actividad metacognitiva de un programador experto puede mostrarse explícitamente a estudiantes de programación por medio de protocolos verbales que aumentados con imágenes.

El uso de Ejemplos Resueltos como método instruccional es eficaz para la adquisición de habilidades de solución de problemas, en dominios estructurados como la programación (Gerjets, Scheiter, y Catrambone 2004; Renkl, Hilbert, y Schworm 2009; Renkl, Atkinson, y Große 2004; M. Ward y J. Sweller 1990; J. Sweller y G. Cooper 1985; Moreno 2006), sobre todo para estudiantes principiantes. (J. R. Anderson, Fincham, y Douglass 1997; VanLehn 1996). Es deseable que los ejemplos resueltos incluyan las características críticas y aspectos clave de los mismos, (J. R. Anderson, C. Boyle, y Corbell 1990).

Los Problemas por Completar son Ejemplos Resueltos con soluciones parciales. Se pueden usar Problemas por Completar para a un tiempo evitar que se inhiba la generación de soluciones nuevas y creativas en el estudiante/programador novato y se fomente un estudio más detallado de los

Ejemplos Resueltos, (van Merrienboer y Krammer 1990; van Merrienboer y Krammer 1987).

Nuevamente, derivadas de la síntesis anterior, e integrando los elementos de los conceptos relacionados con la estrategia instruccional de los Ejemplos Resueltos y los Problemas por Completar, se plantean las proposiciones de la Tabla 6.

Tabla 6. Proposiciones sobre ejemplos resueltos y problemas por completar.

Proposiciones sobre Ejemplos Resueltos y Problemas por Completar	
PERPC1	Un estudiante de programación puede visualizar y estudiar un protocolo verbal como un ejemplo resuelto
PERPC2	La transcripción de un protocolo verbal puede interpretarse como una forma de ejemplo resuelto, cuando el primero es obtenido de un programador experto que resuelve un problema de programación.
PERPC3	Los ejemplos resueltos deben tener descripciones de los elementos clave de su proceso de solución y de la categoría de problema al que pertenecen.
PERPC4	Un protocolo verbal puede ser adaptado como problema por completar si algunos de sus segmentos son adicionados con preguntas a contestar por el estudiante.

Sintetizando los tres grupos de proposiciones discutidos arriba, se tiene que un Sistema Visor de Protocolos Verbales (SVP) es aquel que permite registrar protocolos verbales de solución de problemas de programación de un programador experto, para que un estudiante de programación pueda visualizar explícitamente sus modelos mentales, conocimiento estratégico y actividad metacognitiva. Estos protocolos verbales son aumentados mediante imágenes y adaptados como ejemplos resueltos y problemas por completar.

Por lo tanto, redactados en forma prescriptiva (es decir, obligatoria), los Meta-requerimientos de un SVP son los indicados en la tabla Tabla 7.

Tabla 7. Meta-requerimientos de un SVP

	Meta requerimientos para un SVP
MR1	Un SVP debe permitir registrar protocolos verbales de solución de problemas de programadores expertos.
MR2	Un SVP debe permitir registrar de distintos tipos de problemas de programación e incluir información descriptiva acerca de ellos.
MR3	Un SVP debe permitir a un estudiante visualizar protocolos verbales de solución de problemas de programación, presentados como ejemplos resueltos o problemas por completar.
MR4	Un SVP debe permitir que cada segmento de texto de cada protocolo verbal incluya gráficos o imágenes explicativas.
MR5	Un SVP debe permitir que en algunos tipos de problema, y en ciertos segmentos de su correspondiente protocolo verbal, se inserten preguntas a ser completadas, para presentar el protocolo verbal como un problema por completar.
MR6	Un SVP debe permitir que en ciertos segmentos seleccionados del protocolo verbal se inserten comentarios que se consideren clave para su comprensión.

El Meta-requerimiento MR1, combina las proposiciones sobre protocolos verbales PPV1 a PPV3 y la proposición sobre ejemplos resueltos y problemas por completar PERPC1. El Meta-requerimiento MR2 cubre la proposición PERPC2. El Meta-requerimiento MR3 combina las proposiciones PPV1 a PPV3 y PERPC1 a PERPC3. El Meta-requerimiento MR4 cubre las proposiciones sobre codificación dual PCD1 y PCD2. El Meta-requerimiento MR5 cubre la proposición PERPC4. Los Meta-requerimientos MR2 y MR6 cubren la proposición PERPC3.

3.1.3 Meta Diseño

El siguiente componente de una ISDT para Sistemas Visualizadores de Protocolos Verbales es la identificación de las características ó Meta-diseño, que esta clase de artefacto de software debe tener (ver Tabla 8). Estas características

describen los aspectos funcionales con los que debe contar un SVP. Se derivan de los Meta-requerimientos planteados en la sección anterior.

Tabla 8. Meta-diseño de un SVP

	Meta diseño para un SVP
MD1	Registro y edición de protocolos verbales como secuencias de verbalizaciones.
MD2	Registro de problemas de programación de acuerdo a un tipo o categoría.
MD3	Asociación de uno o varios protocolos a un problema de programación.
MD4	Registro de autores (solucionadores de problemas).
MD5	Asociación de un protocolo a un autor.
MD6	Búsqueda de protocolos asociados a problemas, de acuerdo a tipos o categorías descriptivas
MD7	Visualización y estudio de protocolos verbales mediante un sistema amigable de navegación
MD8	Asociación de imágenes/video de segmentos de código fuente que correspondan a cada segmento de un protocolo verbal.
MD9	Insertar y editar preguntas por completar en ciertos segmentos seleccionados de un protocolo.
MD10	Algunos segmentos de los protocolos verbales registrados en el SVP deben permitir incluir observaciones del instructor (características, complejidad, punto focal, etc.)

Las características de diseño MD1 a MD5 se derivan del Meta-requerimiento MR1. Es decir, para poder registrar un protocolo verbal de la solución de un problema obtenido de un programador experto, un SVP debería ser capaz de registrar por separado tanto la información del autor del protocolo, el problema de programación (y sus características) asociado al protocolo, además del propio protocolo verbal. El registro del protocolo verbal debe hacerse como una secuencia de verbalizaciones o segmentos. Las características de Meta-diseño MD2 y MD6 tienen que ver directamente con el Meta-requerimiento MR2.

La característica MD6 es necesaria para que un estudiante de programación pueda consultar un protocolo verbal (asociado a su vez con un problema) es necesaria una característica de búsqueda que puede ser por tipo de problema, autor o palabras clave.

El Meta-diseño MD7 está asociado al Meta-requerimiento MR3. El MD7 es probablemente la característica de diseño más importante porque es la que permite al estudiante visualizar y explorar el protocolo verbal. Debe ponerse especial cuidado en que esta característica sea fácil de usar.

El Meta-diseño MD8 tiene que ver con el Meta-requerimiento MR4. MD8 tiene que ver con que un SVP permita asociar o adjuntar secuencias de imágenes (que bien pueden ser fragmentos de video) a cada segmento del protocolo verbal. La característica de diseño MD9 cubre el Meta-requerimiento MR5. MD9 tiene que ver con que el SVP permita que un usuario, al revisar un protocolo verbal, sea capaz de seleccionar en algún segmento del protocolo e insertar ahí un pequeño cuestionario para “completar” el protocolo y seguir avanzando hacia la solución y visualización del protocolo.

Finalmente, el Meta-diseño MD10 se asocia al Meta-requerimiento MR6 y tiene que ver con permitir hacer anotaciones sobre cuales partes de la solución son los puntos focales del programa y sobre los que el estudiante debe poner atención.

En conjunto, las características de diseño MD1 a MD10 describen un tipo de artefacto de software que se hipotetiza cubrirá los Meta-requerimientos descritos en la sección 3.1.2, derivados a su vez de las principales facetas del problema del aprendizaje de la programación.

3.1.4 Hipótesis del producto de diseño

De acuerdo con Walls, et. al. (1992, p.55) los Meta-requerimientos y el Meta-diseño de una Teoría de Diseño de Sistemas de Información deben ser probados empíricamente mediante el desarrollo y uso de un artefacto específico que se derive de la aplicación de tal Teoría de Diseño. La mayoría de estas hipótesis tienen que ver con la *factibilidad* de construir tal clase de artefacto, mientras que otras están relacionadas con la *efectividad* del mismo.

Entonces, para la presente tesis las hipótesis comprobables del producto de diseño de una ISDT para un Sistema Visualizador de Protocolos Verbales (SVP) son las mostradas en la Tabla 9. Las hipótesis PROD1 a PROD4 tienen que ver con probar que un artefacto con las características descritas por el Meta-diseño es de hecho factible de construir. La hipótesis PROD5 se relaciona con la efectividad derivada del uso de tal artefacto, en el sentido de fomentar un aprendizaje significativo y una transferencia de conocimiento en estudiantes en las fases iniciales de aprendizaje de la programación, en el sentido de la definición dada por (Mayer y Wittrock 1996), donde tal transferencia ocurre cuando una persona usa lo aprendido para resolver nuevos problemas.

Tabla 9. Hipótesis comprobables del producto de diseño.

Hipótesis comprobables del producto de diseño	
PROD1	Es factible diseñar un SVP que permita registrar, visualizar y editar protocolos verbales de programadores expertos, aplicados a la solución de problemas de programación.
PROD2	Es factible diseñar un SVP que permita registrar distintas categorías de problemas de programación que se asocien a uno o a varios protocolos verbales.

(…Continuación) Tabla 9. Hipótesis comprobables del producto de diseño

Hipótesis comprobables del producto de diseño	
PROD3	Es factible diseñar un SVP que permita visualizar protocolos verbales aumentados con animaciones de código.
PROD4	Es factible diseñar un SVP que permita complementar segmentos de protocolos verbales con preguntas por completar por los estudiantes.
PROD5	Un SVP diseñado de acuerdo a las hipótesis PROD1 a PROD4 fomentará una transferencia de habilidades de aplicación de estrategias de programación en estudiantes que lo utilicen en sus primeras etapas de aprendizaje.

3.2 Proceso de diseño.

3.2.1 Método de diseño.

Walls et. al. (1992) admiten que hay muchos aspectos importantes que intervienen en el proceso de diseño detallado de cualquier tipo de sistema de información (por ejemplo, diseño de bases de datos, diseño de interfaces, diseño de algoritmos), pero acotan diciendo que "dado que se ha realizado extensiva investigación sobre la efectividad de estos aspectos a lo largo del tiempo, no nos enfocaremos en éstas áreas aquí" (Walls, et.al., 1992, p.55).

Por tanto el planteamiento del componente del proceso de diseño de una ISDT se enfoca en lo que equivale a la etapa de determinación de requerimientos del ciclo de vida de desarrollo de software[15], o también llamado *método de identificación de requerimientos* ó IRDM (Information Requirements Determination Method, por sus siglas en inglés).

[15] Software Development Life Cycle o SDLC por sus siglas en inglés.

3.2.2 Teorías núcleo.

La Ingeniería de Software es rica en métodos de determinación de requerimientos. Desde los métodos clásicos estructurados (Yourdon 1989; Gane y Sarson 1977) basados en el modelado de procesos y flujos de datos, pasando por técnicas orientadas a objetos (Rumbaugh, Jacobson, y Booch 2000; Rumbaugh, Jacobson, y Booch 2004) hasta las estrategias enfocadas en el desarrollo de prototipos (Davis 1992; Tanik y Yeh 1989; Gordon y Bieman 1995; Phalp y Counsell 2001), fuertemente basadas en el uso de los llamados lenguajes de cuarta generación (4GL) y herramientas CASE (Computer Aided Software Engineering), el repertorio de opciones para identificar los requerimientos de un sistema de información es sumamente amplia.

3.2.2.1 Prototipos evolutivos incrementales.

Sommerville (2002) describe a un prototipo de software como “una versión inicial de un sistema de software que se utiliza para demostrar conceptos, probar las opciones de diseño y comprender mejor el problema y sus posibles soluciones”. Experimentos realizados (Boehm y Gray 1984) indican que la construcción de prototipos reduce el número de problemas relacionados con la especificación de requerimientos. Gordon y Bieman (1995) reportan que al estudiar una muestra de treinta y nueve aplicaciones industriales desarrolladas por medio de prototipos, treinta y tres de ellas tuvieron mejoras en la usabilidad del sistema, mejoras en la calidad del diseño, mejoras en el mantenimiento y reducción en el esfuerzo de desarrollo.

Existen dos grandes enfoques para la construcción de prototipos: el prototipo desechable y el prototipo evolutivo (Pressman 2002, p.192; Sommerville 2002, pp.174-175; Davis 1992).

El prototipo desechable tiene como único objetivo ayudar a clarificar y refinar los requerimientos. Éste es evaluado y modificado durante varias iteraciones mediante la retroalimentación del usuario. Esta evaluación da la pauta para *documentar formalmente* los requisitos identificados, después de lo cual el prototipo deja de tener utilidad y se desecha, procediendo después con el proceso de desarrollo mediante otros paradigmas.

El prototipo evolutivo inicia como un sistema sencillo que incluye los requerimientos más importantes del usuario y se va aumentando o cambiando según se descubren nuevos requisitos para finalmente convertirse en el sistema solicitado por el usuario.

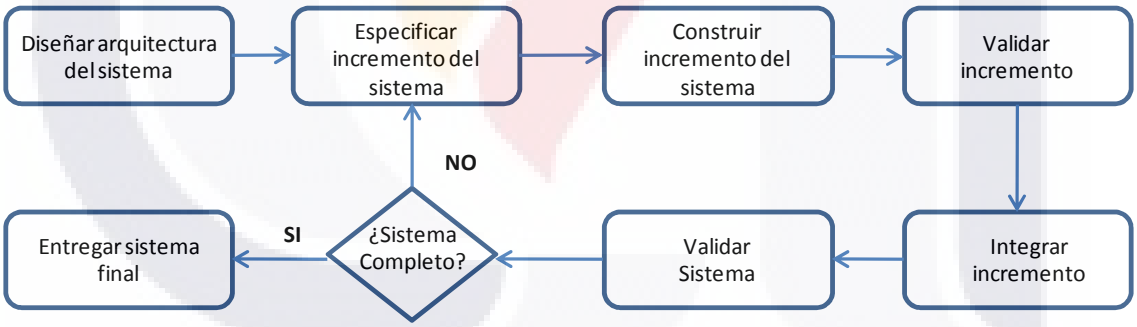


Figura 11. Proceso de construcción de prototipos evolutivos e incrementales (Tomado de Sommerville (2002))

La estrategia de prototipos evolutivos consiste en desarrollar una versión inicial, exponerla a la retroalimentación de los usuarios, depurarla, y después de varias iteraciones, llegar al sistema requerido. La característica fundamental de esta estrategia de desarrollo consiste en que el prototipo no se desecha, sino que se convierte en el sistema final, a través de un proceso de refinamiento iterativo.

El método de prototipos evolutivos comparte muchas semejanzas con estrategias afines tales como RAD (Rapid Application Development) y JAD (Joint Applicaction Development) (referencias).

El proceso de diseño y desarrollo de prototipos evolutivos tiene las siguientes características (Sommerville 2002, p.176).

a) Las actividades de especificación, diseño e implementación están entremezcladas, dado que no existe una especificación detallada del sistema. La documentación de diseño producida depende de las herramientas usadas para implementar el prototipo. El documento de requerimientos solo define las características más importantes del sistema.

   Haciendo una comparación con los componentes de la ISDT obtenidos, vemos que no se cuenta con una especificación *detallada* del sistema (en este caso, la clase de sistema). La redacción de Meta-requerimientos y el Meta-diseño (los objetivos buscados y las características deseadas de la clase de artefacto) se hacen en términos de enunciados precisos, pero no corresponden a un diseño detallado. Es decir, el Meta-diseño lista las características de diseño más importantes deseadas, pero no sus propiedades y comportamiento.

b) El sistema se desarrolla en una serie de incrementos. El usuario final se involucra en el diseño y evaluación de cada incremento. Se pueden proponer cambios en el software (artefacto) y sugerir nuevos requerimientos en versiones posteriores (incrementos) del sistema.

   Esta característica del enfoque de prototipos evolutivos se adapta a las características de la ISDT para SVPs, en donde puede seguirse un diseño y desarrollos por incrementos (por ejemplo, en un incremento, la edición de protocolos, en el siguiente incremento la visualización, en el siguiente la

visualización de problemas por completar, etc.) y en donde los estudiantes pueden sugerir mejoras en las características del artefacto desarrollado.

c) Se usan técnicas de desarrollo rápido tales como el uso de herramientas CASE (Computer Aided Software Engineering) y lenguajes de cuarta generación (4GL)

La implementación de un artefacto que se derive del Meta-diseño descrito en la ISDT para SVPs puede beneficiarse de este tipo de herramientas, en donde la capacidad de generar código automáticamente y diseñar, probar e integrar modelos de datos con interfaces de usuarios rápidamente, pueden substancialmente acelerar el proceso general de diseño y mejorar su calidad[16].

d) La interfaz de usuario se desarrolla mediante herramientas y entornos de desarrollo visual, en donde el diseñador construye el sistema de información de manera interactiva, siguiendo la metáfora de arrastrar y soltar, tomando componentes predefinidos como botones, cajas de texto y menús. Al respecto, Ousterhout (1998) encontró que el uso de entornos de desarrollo visual reduce radicalmente el tiempo de desarrollo de un sistema de información.

En relación a la ISDT propuesta, este punto resulta de especial importancia como componente del método de diseño, ya que la característica funcional más importante de la clase de artefacto hipotetizada, es su capacidad de *visualizar protocolos verbales* en forma de ejemplos resueltos y problemas por completar, por lo que la capacidad de diseñar, modificar y adaptar eficientemente una interfaz de usuario adecuada es crítica.

[16] Mención especial merecen las herramientas de captura de video, como se comentará en el siguiente capítulo.

Como argumento adicional para guiar la selección de la estrategia de prototipeo adecuada, Andriole (1992) sugiere evaluar un conjunto de seis preguntas básicas, que se enumeran en la Tabla 10

Tabla 10. Selección de la estrategia de prototipeo adecuada (Tomado de Pressman (2002), adaptado de Andriole (1992)

Pregunta	Prototipo desechable	Prototipo Evolutivo	Trabajo Preliminar adicional requerido
¿Se entiende el dominio de la aplicación?	SI	SI	NO
¿Se puede modelar el problema?	SI	SI	NO
¿Está el usuario suficientemente seguro de los requisitos básicos del sistema?	SI/NO	SI	NO
¿Están establecidos los requisitos y son estables?	NO	SI	SI
¿Hay requisitos ambiguos?	SI	NO	SI
¿Hay contradicciones en los requisitos?	SI	NO	SI

Así, tenemos que en el planteamiento de la ISDT para SVPs:

a) Se conoce el dominio de la aplicación (el aprendizaje de la programación).
b) Es posible modelar el problema (mediante técnicas de ingeniería de software, tales como casos de uso),
c) No se cuenta con un usuario o grupo de usuarios particulares, (en realidad se hablaría de una *clase de usuarios* quienes son estudiantes de programación básica), en cuyo caso este planteamiento no aplica,
d) Los requisitos están definidos y son estables (es decir, los Meta-requerimientos de la ISDT para SVPs, ver Tabla 7),

e) No hay requisitos ambiguos, en donde el Meta-diseño (ver Tabla 8) indica con suficiente claridad las características deseadas de la clase de artefacto requerido y
f) No existen contradicciones aparentes en los Meta-requerimientos.

Por lo anterior, se argumenta que el método de prototipos evolutivos puede ser adecuado como método de diseño de una ISDT para SVPs, en el sentido de permitir refinar los requerimientos detallados de diseño, reducir el tiempo y costo de desarrollo, permitir un diseño e implantación ágil de la interfaz de usuario y fomentar el uso de herramientas de desarrollo rápido tales como CASE y 4GL.

3.2.3 Hipótesis del proceso de diseño

El componente final de la ISDT para Sistemas Visualizadores de Protocolos Verbales (SVP's) son las hipótesis de proceso de diseño. De acuerdo con Walls, et. al. (1992) éstas tienen la función de permitir verificar si el aplicar el método de diseño resulta en un artefacto que sea consistente con el meta-diseño. Dado que el método de diseño descrito en la sección anterior se basa en Prototipos Evolutivos, el planteamiento de la hipótesis de proceso de diseño (ver Tabla 11) se redacta en términos de verificar la factibilidad de producir un artefacto con las características descritas por su Meta-diseño, por medio de dicho método.

Tabla 11. Hipótesis de proceso de diseño de un SVP.

Hipótesis comprobables del proceso de diseño de Sistemas Visores de Protocolos Verbales	
PROC1	Es factible diseñar SVPs por medio de la estrategia de prototipos evolutivos incrementales.

De tal suerte, en el siguiente capítulo se describe el proceso de desarrollo de un Sistema Visor de protocolos, para efecto de comprobar las hipótesis de producto y proceso de diseño

4 DESARROLLO DE UN SISTEMA VISOR DE PROTOCOLOS VERBALES.

Un diseñador sabe que ha alcanzado la perfección, no cuando ya no hay nada más que agregar, sino cuando ya no hay nada más que quitar.

- Antoine de Saint-Exupery, Escritor y Aviador (1900-1945).

De acuerdo con Hevner et. al. (2004) el resultado de una investigación en el campo de las ciencias del diseño y en el área de los sistemas de información es, por definición, un artefacto de TI creado para atender un problema relevante. Proponen que el principio número uno para conducir una investigación en las ciencias del diseño es *producir un artefacto viable*, bajo la forma de un constructo, modelo, método, o instanciación. El concepto de instanciación consiste precisamente en construir un artefacto derivado de principios de diseño explícitos (Weber 1987).

Pero Hevner et. al. (2004) acotan lo anterior indicando que "los artefactos instanciados en el proceso de investigación en las ciencias del diseño, raramente son sistemas de información completamente maduros. En su lugar, estos artefactos son innovaciones que definen las ideas, prácticas, habilidades técnicas y productos a través de los cuales el análisis, diseño, implementación y uso de sistemas de información pueden ser efectiva y eficientemente alcanzados". De tal suerte que en el ámbito de las ciencias del diseño debe entenderse al artefacto más como un medio que como un fin; en este caso, el medio para lograr una representación precisa del problema de investigación. En este sentido, otros investigadores han observado que "resolver un problema sencillamente significa representarlo de tal manera que su solución se vuelva transparente" (Simon 1996, p.132).

4.1 Desarrollo del artefacto.

El punto de enfoque para guiar la instanciación de un artefacto es su Meta-diseño. El Meta-diseño de una ISDT para Sistemas Visores de Protocolos se muestra en la Tabla 8. Dado que el método de diseño propuesto para una ISDT para Sistemas Visores de Protocolos es el modelo de desarrollo por prototipos evolutivos (ver sección 3.2), la primera actividad sugerida para iniciar la creación del artefacto es la definición de la arquitectura del sistema. Realizar este diseño arquitectónico sirve para identificar cuales secciones o partes (es decir, incrementos) del sistema se desarrollarán primero (ver Figura 12).

4.1.1 Arquitectura.

Pressman (2002) indica que el diseño arquitectónico es la representación de la estructura de los datos y componentes que se requieren para construirlo, de forma que el resultado de esta actividad debería consistir de una representación estructural que muestre los componentes principales del artefacto, y sus interacciones. Para este proceso de instanciación se optó por esquematizar la arquitectura desde un punto de vista funcional, por medio de casos de uso (Rumbaugh, Jacobson, y Booch 2000; Rumbaugh, Jacobson, y Booch 2004) en donde cada caso de uso representa una pieza de funcionalidad requerida del sistema, tomando como guía el Meta-diseño de la ISDT para SVP (Tabla 8)

Desde una perspectiva funcional el artefacto consta de dos subsistemas. El primero es un Subsistema de Edición, en donde un usuario instructor debería poder registrar, editar y borrar protocolos verbales, problemas de programación y autores de protocolos. La edición de los protocolos requiere a su vez opciones adicionales de asociación a autores y problemas, asociación de imágenes e

inserción de preguntas por completar en determinados segmentos de un protocolo elegidos por un instructor (ver Figura 12[17]).

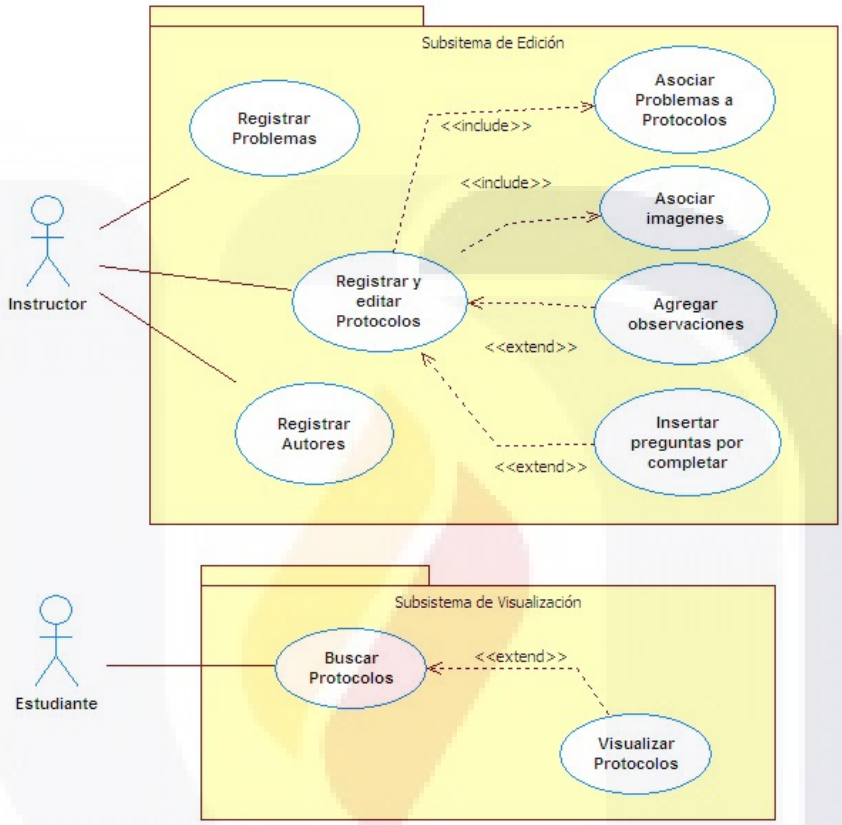


Figura 12. Arquitectura funcional del artefacto visor de protocolos verbales.

El otro componente del artefacto es el Subsistema de Visualización, en donde un usuario estudiante podría consultar un protocolo de solución de acuerdo a distintos atributos y después poder visualizar explícitamente uno o varios

[17] La figura 12 no representa, por supuesto, la arquitectura del prototipo en el sentido clásico de Ingeniería de Software, en donde suelen utilizarse otro tipo de diagramas, tales como diagramas de clase o diagramas de estructura. Sin embargo, para fines prácticos, el diagrama de casos de uso cumple con el objetivo de mostrar una representación de alto nivel de abstracción de las partes que compondrían el prototipo completo, desde una perspectiva funcional.

protocolos verbales asociados a tal problema. Es en este último componente (caso de uso) en donde los principios teóricos de las teorías núcleo de visualización (3.1.1.1) y de diseño instruccional (3.1.1.3) deben implementarse, poniendo por lo tanto especial atención al diseño de la interfaz de usuario.

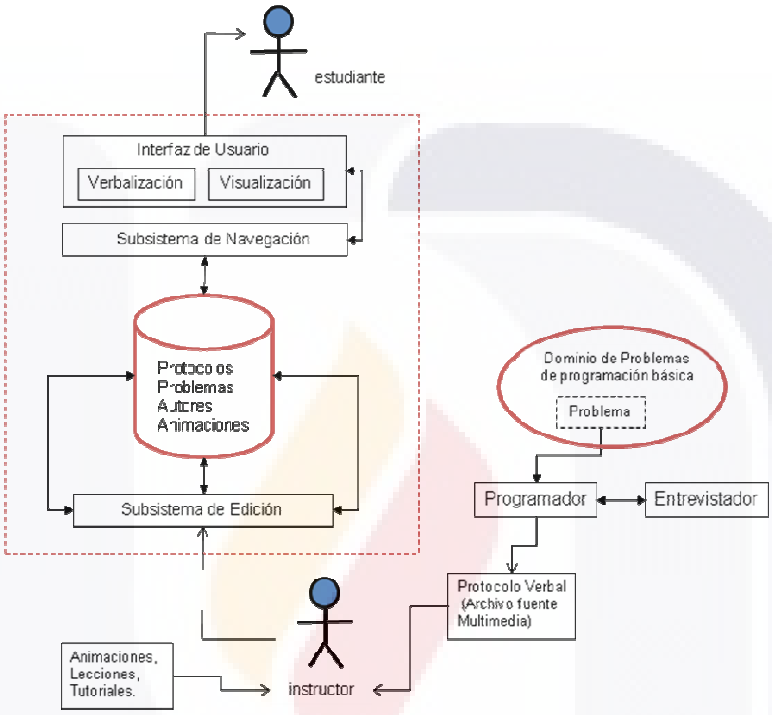


Figura 13. Perspectiva arquitectónica complementaria.

Otra perspectiva arquitectónica del artefacto puede apreciarse en la Figura 13, en donde se entiende que existe un componente que contiene la información relacionada con los protocolos, problemas de programación, animaciones e información de los autores de los protocolos. Este componente básico es de hecho la base de datos y su diseño también merece también atención especial (ver sección 4.1.2)

Es importante observar aquí que fuera del sistema se realiza el proceso de registro de protocolos, en donde un entrevistador y un programador experto interactúan siguiendo los principios indicados por Ericsson y Simon (1993), tal

como se indica en la sección 3.1.1.2. Este proceso genera como salida un registro audiovisual (es decir, un archivo de video) que serviría de entrada para el subsistema de edición del artefacto. El detalle de este proceso se comenta también en la sección 4.1.2.

El método de diseño por prototipos evolutivos indica que dadas varias iteraciones (ciclos) debe especificarse el incremento del sistema que se implementará. Para el caso de la instanciación del artefacto Visor de Protocolos y dada la arquitectura mostrada en las Figuras 12 y 13, se optó por que el primer incremento fuese el Subsistema de Visualización.

4.1.2 Modelo de datos.

En 1976 Peter Chen (Chen 1976), propuso la técnica de representación conceptual de datos llamada Entidad-Relación (E-R). En la Ingeniería de Software un modelo E-R sirve para obtener una representación abstracta de los datos de un sistema de información, en términos de objetos (es decir, entidades) y sus relaciones. La modelación E-R es un método para producir un modelo semántico que a su vez sirve de entrada para construir una base de datos relacional. El tema de la modelación de bases de datos relacionales es uno de los mejor documentados en la Ingeniería de Software (Date 1990; Elmasri y Shamkant 2000; Rob, Coronel, y Crocket 2008) existiendo además una gran variedad de herramientas para facilitar su aplicación.

Para la instanciación del artefacto, un requerimiento central consiste en obtener una representación de datos que permita registrar protocolos verbales para su posterior edición y visualización. Por ejemplo, en una sesión de registro de protocolos se solicitó a un programador experto que resolviera el siguiente problema:

Escribir un programa en Lenguaje C que al recibir como dato N números enteros obtenga la suma de los números pares y el promedio de los impares.

De acuerdo con el proceso recomendado por Ericsson y Simon (1993), el primer paso consistió en la transcripción del protocolo. La transcripción literal de una parte del protocolo resultante de un sujeto tratando de resolver el problema anterior, puede verse en la Tabla 12:

Tabla 12. Transcripción literal de un protocolo verbal.

Primero, necesito saber cuántos números enteros voy a capturar. Necesito un contador, para guardar la cantidad de números enteros y saber en qué momento voy a dejar de capturarlos. Un acumulador para la suma y una variable para guardar el promedio de los números. …pero no, espérame, otra forma de solucionar el problema es… Primero identificar las variables. Entonces, tengo la variable N, que sería la cantidad de valores enteros que necesito capturar. Una variable para almacenar la suma de los números pares, y otra para almacenar el promedio de los impares. Entonces, lo primero que se hace es definir las librerías del programa. La primer librería va a ser el STDIO, porque aquí se encuentran almacenadas las funciones principales [de entrada y de salida] que vienen siendo el printf y el scanf. Otra librería que podemos usar es la CONIO, porque aquí están almacenadas las instrucciones que le indican al sistema operativo que limpie la pantalla o detenga la pantalla para poder ver resultados. Después se declara la función main que es la encargada de realizar el proceso o ejecución del programa. Dentro de la función main declaramos las variables..

Puede observarse que bajo esta forma el protocolo verbal es muy semejante a un monólogo que podría resultar poco útil para un aprendiz de programación, ya que implicaría una alta carga cognitiva por la gran cantidad de información que contiene.

Siguiendo el proceso sugerido por Ericsson y Simon, el paso siguiente en la transcripción del protocolo fue la segmentación del mismo, en donde cada verbalización constituía una instancia o paso de un proceso general. Siguiendo el método, se observó que para realizar la segmentación, algunas pistas apropiadas podían ser las pausas, entonaciones o marcadores gramaticales emitidos por el sujeto.

Sin embargo, al llegar a este punto (es decir, la segmentación), se optó por complementar la técnica propuesta por Ericcson y Simon (1993) agregando un elemento adicional: además de registrar la información verbal del sujeto (es decir, la secuencia de oraciones) se registró también la información visual del proceso de solución. Esto es, se utilizó una herramienta de captura de video para registrar cómo el programador experto escribía el programa al tiempo que pensaba en voz alta. Este tipo de herramientas permitieron grabar en forma de video toda la actividad que sucedía en la pantalla de la computadora, cuando el programador experto estaba interactuando con ella.

Así, en este caso el producto generado por una sesión de registro de protocolos no fue una cinta o archivo de audio, sino un archivo de video que contenía tanto la voz del autor del protocolo, como las acciones realizadas por éste al escribir el programa.

Esta decisión proporcionó dos grandes beneficios: servir de referencia para el diseño posterior de secuencias de video que fomentasen la codificación dual, y servir de guía para la segmentación del protocolo verbal, en donde al analizar el video generado, se podía observar la correspondencia entre la verbalización y el código escrito del programa en la secuencia de video[18].

Una vez segmentado, un ejemplo de la transcripción de un protocolo verbal, tomado de una sesión de resolución de problemas de programación, toma la forma indicada en la Tabla 13.

[18] Esta correspondencia no siempre resultó evidente: en ocasiones el sujeto verbaliza más rápido de lo que escribe o viceversa.

Tabla 13. Ejemplo de un protocolo verbal segmentado.

Verbalización
Primero, necesito saber cuántos números enteros voy a capturar.
Necesito un contador, para guardar la cantidad de números enteros y saber en qué momento voy a dejar de capturarlos.
Un acumulador para la suma y una variable para guardar el promedio de los números.
…pero no, espérame..otra forma de solucionar el problema es…
Primero identificar las variables.
Tengo la variable N, que sería la cantidad de valores enteros que necesito capturar.
Una variable para almacenar la suma de los números pares
Y otra para almacenar el promedio de los impares.
Entonces, lo primero que se hace es definir las librerías del programa
La primer librería va a ser el STDIO
Porque aquí se encuentran almacenadas las funciones principales [de entrada y de salida] que vienen siendo el printf y el scanf
Otra librería que podemos usar es la CONIO, porque aquí están almacenadas las instrucciones que le indican al sistema operativo que limpie la pantalla o detenga la pantalla para poder ver resultados
Después se declara la función main que es la encargada de realizar el proceso o ejecución del programa
Dentro de la función main declaramos las variables.

Con esta estructura básica, la información del protocolo verbal ya es susceptible de ser representada en un modelo E-R por lo que teniendo nuevamente como referencia los elementos del Meta-diseño de nuestra ISDT (Tabla 8, elementos MD1 a MD10), los requerimientos de información pueden replantearse de la siguiente manera:

1. Registrar información de los autores (programadores expertos) de los protocolos.
2. Registrar información de los problemas de programación a resolver, la cual incluye el texto del problema y características tales como su categoría, complejidad y palabras clave, entre otras.

3. Registrar protocolos verbales segmentados, en donde cada segmento cuenta con imágenes (video) asociado. Un autor puede resolver varios protocolos. Un problema puede también tener asociados varios protocolos (es decir, varios expertos pueden solucionar un mismo problema siguiendo distintas estrategias de solución).
4. Registrar tanto el texto como una imagen de video en cada segmento de un protocolo. Un segmento del protocolo puede ser un "punto focal", de acuerdo al concepto definido por Rist (1995).
5. Un segmento de un protocolo puede requerir la solución de un "Problema por completar", de acuerdo a lo recomendado por van Merrienboer y Krammer (1987, 1990), para continuar la visualización.

El tercer y último paso sugerido por Ericsson y Simon es la codificación del protocolo. La sugerencia en este punto es el diseño de un esquema de codificación a-priori para facilitar el análisis del protocolo. En el sentido clásico, el propósito de Ericsson y Simon acerca de esta actividad de codificación es facilitar el análisis cognitivo por medio de un marco teórico, principalmente para la identificación de patrones de comportamiento y de un vocabulario común al dominio del problema que se está analizando.

Pero para la instanciación del artefacto, la interpretación de esta actividad de codificación se entendió como parte del diseño del modelo E-R, en donde de manera natural algunos 'códigos' se asignan a segmentos de un protocolo. Por ejemplo: a) cada segmento debe tener una identificación (ID), b) todos los segmentos deben tener un número de secuencia, c) algunos segmentos corresponden a 'puntos focales' del proceso de resolución del problema, c) todos los segmentos deben contar con un correspondiente 'código visual' o animación, entendida bajo los principios de la Teoría de Codificación Dual (ver sección 3.1.1.1).

Dados todos los requerimientos anteriores, el modelo E-R resultante es el mostrado en la Figura 14[19]. Los problemas, autores, protocolos, verbalizaciones (segmentos) y segmentos (problemas) por completar se representan como entidades. Las relaciones entre ellas pueden leerse de la siguiente manera:

1. Un problema puede tener asociados uno o más protocolos.
2. Un autor puede resolver uno o más protocolos.
3. Un protocolo está asociado a solo un problema.
4. Un protocolo tiene un solo autor.
5. Un protocolo consta de una o más verbalizaciones.
6. Cada verbalización puede tener uno y solo un problema por completar.

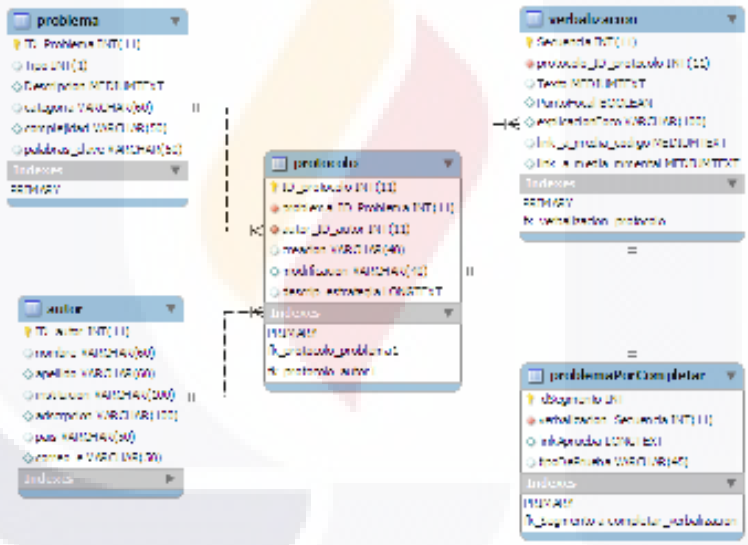


Figura 14. Modelo E-R propuesto para la instanciación del artefacto.

Una vez diseñado el modelo de datos, se continuó con el proceso de instanciación del artefacto, procediendo al diseño de la interfaz de usuario, procurando lograr una funcionalidad que fuese consistente con el Meta-diseño de la ISDT (Tabla 8). Este proceso se discute en la siguiente sección.

[19] Par un mayor detalle sobre las reglas de modelación E-R consultar, por ejemplo, (Date 1990; Chen 1976; Elmasri y Shamkant 2000)

4.1.3 Interfaz de usuario.

La interfaz de usuario es la parte del software visible para el usuario y es uno de los aspectos más importantes a diseñar correctamente. De hecho, un mal diseño en este componente puede provocar que los usuarios sencillamente descarten el uso del software. En la literatura de Ingeniería de Software existen numerosos principios para guiar el diseño de interfaces de usuario. Entre otros, pueden mencionarse las “reglas de oro” de diseño de interfaces recomendadas por Mandel (1997):

a) Dar el control al usuario. En este punto se comentan recomendaciones generales en el sentido de facilitar la interacción del usuario con la interface, tales como definir modos de interacción que no obliguen al usuario a realizar acciones innecesarias y no deseadas, permitir que la interacción se pueda interrumpir y deshacer, aligerar la interacción a medida que avanza el nivel de conocimiento y permitir personalizar la interacción.
b) Reducir la carga de memoria del usuario. En este aspecto, se sugiere que la interfaz reduzca la demanda de memoria de corto plazo del usuario evitando forzar el recuerdo constante de acciones anteriores para poder avanzar, incluyendo claves visuales para que el usuario reconozca acciones anteriores sin tenerlas que recordar, estableciendo valores por defecto útiles, creando formatos basados en metáforas del mundo real y/o desglosando la información en forma progresiva.
c) Construir una interfaz consistente. Las recomendaciones sobre este aspecto tienen que ver con procurar que todos los mecanismos de entrada se limiten a un conjunto limitado y que sean los mismos en toda la aplicación, en permitir que el usuario realice las tareas en el contexto adecuado, es decir incluyendo indicadores que posibiliten al usuario conocer el contexto del sistema en el que se encuentra, que permitan al

usuario saber de qué lugar procede y que alternativas existen para la transición a una tarea nueva.

Existen otros principios para guíar el diseño de interfaces de usuario en la literatura de ingeniería de software, con distintos niveles de detalle (Pressman 2002; Sommerville 2002; Jakob Nielsen 1993; Wells 2006; Knox, Bailey, y Lynch 1989), pero todos coinciden en que de una u otra manera, las interfaces deben ser sometidas a prueba por una población de usuarios igual o similar al usuario final.

El diseño de la interfaz de usuario debería cumplir con la descripción del Meta-diseño MD7 (Visualización y Estudio de protocolos verbales mediante un sistema amigable de navegación) y MD8 (Asociación de imágenes/video de segmentos de código fuente que correspondan a cada segmento de un protocolo verbal).

La primera aproximación de diseño hacia una interfaz funcional (ver Figura 15) se realizó con una herramienta de animación[20], buscando que la inclusión de ayudas visuales al usuario (tales como uso de colores, parpadeos intermitentes, animaciones, flechas para indicar al usuario a donde enfocarse visualmente) pudieran ayudar en la retención del proceso de solución del problema de programación, enfatizando aspectos de codificación dual y disminución de carga en la memoria de corto plazo.

En esta primera aproximación se obtuvo información importante del proceso de construcción, obtenida de los propios desarrolladores. Es decir, se detectaron estos temas, aún antes de iniciar pruebas con usuarios reales:

[20] Macromedia Flash, version 8.

1. La curva de aprendizaje para la utilización de la herramienta de animación fue larga y el tiempo de producción de una versión funcional fue también muy extensa.
2. La interface debería ser simplificada para reducir la carga cognitiva, eliminando algunos elementos, tales como ventanas y flechas.
3. La idea de navegación con íconos representando “adelante y atrás” fue funcional y se conservó.
4. La herramienta de animación no se prestaba para conectarse con un manejador de bases de datos, que contuviese la información de los protocolos de solución de problemas.
5. La ejecución del protocolo requería instalar componentes adicionales en la máquina del usuario[21].

De los puntos anteriores, el factor definitivo para descartar esta estrategia de diseño (uso de herramientas de animación), fue el punto cuatro, ya que cada protocolo debía construirse a razón de uno a uno y la integración con un manejador de base de datos resultaba técnicamente muy compleja. Esta situación hacía potencialmente poco eficaz el proceso de producción.

[21] Macromedia Flash Player.

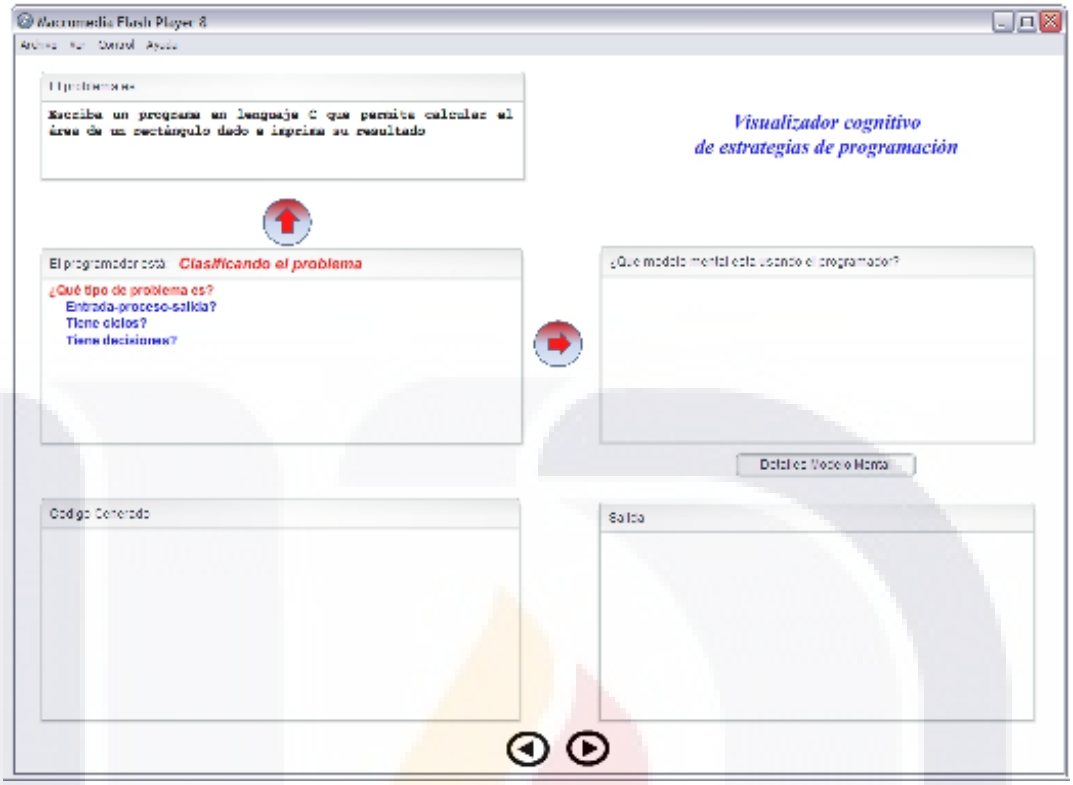


Figura 15. Primera versión de interfaz de usuario del visor de protocolos construida con Macromedia Flash.

Dado que el primer prototipo animado no tenía la funcionalidad de conexión a una base de datos, se optó por diseñar un nuevo prototipo con una herramienta 4GL[22] (Sommerville 2002, p.183), con características de bases de datos y diseño rápido de pantallas. Este prototipo (ver Figura 16) ya contaba con el modelo de datos diseñado (ver Figura 14) y se construyó también solamente con la finalidad de obtener información técnica del proceso de desarrollo.

Como en el caso de la interfaz animada, esta aproximación no fue sometida a uso por parte de usuarios representativos de la población final y se desarrolló para obtener información técnica para observar cómo se comportaba el registro de información de un protocolo en el modelo de datos diseñado y evaluar alternativas de visualización para la animación/visualización del proceso de escritura de

[22] Microsoft Access 2003.

programas que previamente se había grabado en los protocolos audiovisuales (ver sección 4.1.2).

Se observó que la herramienta 4GL utilizada no soportaba la asociación de videos que acompañaran a cada segmento del protocolo. La única manera de mostrar una imagen era por medio de gráficos animados, de los cuales el tiempo y esfuerzo de desarrollo (una secuencia animada para cada verbalización) era prohibitivo. Sin embargo, se comprobó la factibilidad de construir, registrar y visualizar la información en una base de datos generada a partir del modelo de datos diseñado (Figura 14).

La experiencia técnica obtenida con esta segunda aproximación, señaló la conveniencia de utilizar tecnología Web como plataforma de desarrollo, debido a las prestaciones en cuanto al uso de bases de datos, consistencia en estándares para desarrollo de interfaces, disponibilidad de herramientas de desarrollo rápido (RAD) (Andriole 1992; Tanik y Yeh 1989; Gordon y Bieman 1995) y facilidades para despliegue de contenido multimedia (Oviatt 2006; Chansilp 2004).

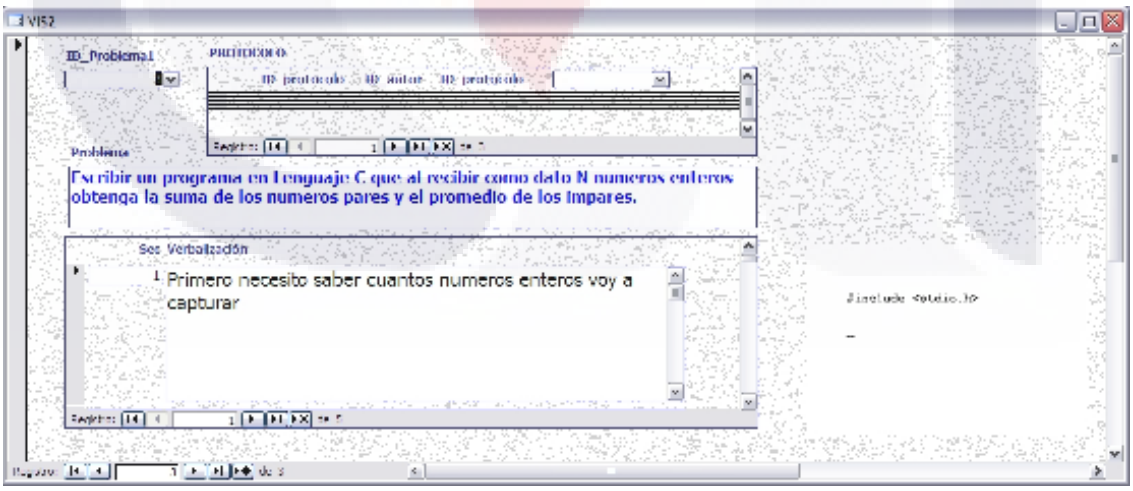


Figura 16. Interfaz de usuario prototipo en una herramienta 4GL (Access)

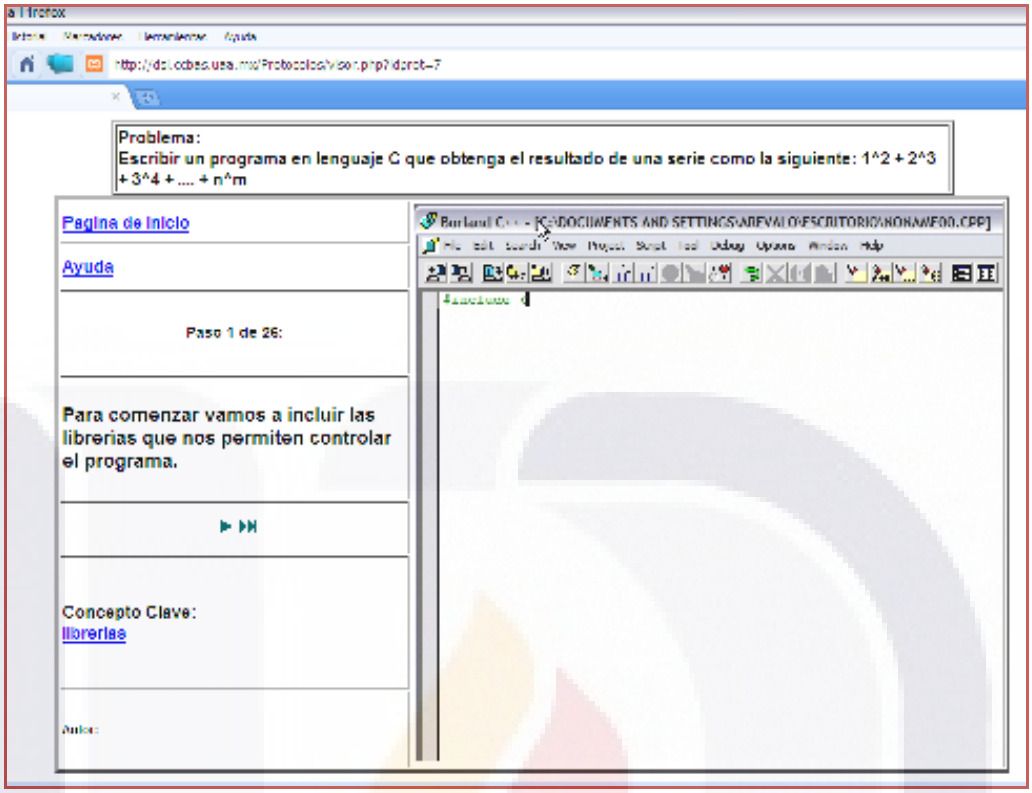


Figura 17. Interfaz de usuario del prototipo basada en Web.

Así pues, para desarrollar el primer prototipo funcional (ver Figura 17) se recurrió a una plataforma de desarrollo basada en Web, con los siguientes componentes: un servidor de bases de datos MySQL, el lenguaje de programación PHP para desarrollo de las páginas y una herramienta RAD para entornos Web[23]. Se utilizó nuevamente una herramienta de captura de video para producir las secuencias de video que simularan la escritura de código asociada al protocolo verbal[24].

En el diseño de la interfaz se procuró respetar los principios de codificación dual sugeridos en la correspondiente teoría núcleo de la ISDT para SVP, mostrando en la parte izquierda de la pantalla la información textual (código verbal) y en la parte derecha las imágenes correspondientes a la animación del

[23] Macromedia Dreamweaver.
[24] Techsmith Camtasia Studio.

código (código visual), de tal suerte que se buscara obtener una mejor retención en la memoria de largo plazo del estudiante, tanto de un modelo mental específico (por ejemplo “como saber que un número es par o impar”), como de una estrategia general de solución (por ejemplo “dentro de un ciclo capturar los N números enteros y sumar los pares y promediar los impares usando la operación aritmética del residuo de una división para distinguirlos).

Bajo esta forma, se cargaron cuatro protocolos verbales en el prototipo, tomados de diversos profesores voluntarios de programación básica. Al contarse con esta versión funcional del prototipo, se procedió a realizar pruebas de usabilidad con usuarios reales.

4.2 Evaluación del artefacto.

La utilidad, calidad y eficacia de un artefacto diseñado deben demostrarse rigurosamente por medio de métodos de evaluación bien ejecutados. Hevner, March, y Park (2004, p.86) enfatizan que la evaluación es un componente crucial del proceso de investigación en las ciencias del diseño y proporcionan un compendio de métodos (Tabla 14), señalando que éstos deben seleccionarse apropiadamente, de acuerdo con las características y objetivos del propio artefacto.

Tabla 14. Métodos de evaluación de artefactos. Tomado de Hevner, et. al. (2004).

Por observación	Caso de estudio: Estudiar el artefacto a profundidad en el entorno de negocios
	Estudio de campo: Monitorear el uso del artefacto en múltiples proyectos.

(…Continuación) Tabla 14. Métodos de evaluación de artefactos. Tomado de Hevner, et. al. (2004)

Analíticos	Análisis Estático: Examinar cualidades estáticas de la estructura del artefacto (por ejemplo, complejidad)
	Análisis de Arquitectura: Estudio de ajuste del artefacto hacia una arquitectura de sistemas de información.
	Optimización: Demostrar propiedades inherentemente óptimas del artefacto o proveer límites de optimalidad en el comportamiento del artefacto.
	Análisis Dinámico: Examinar cualidades dinámicas en el uso del artefacto (por ejemplo, rendimiento)
Experimentales	Experimento controlado: Estudiar cualidades el artefacto en un entorno controlado (por ejemplo, usabilidad)
	Simulación: ejecutar el artefacto con datos artificiales.
Pruebas	Prueba Funcional (caja negra): Ejecutar interfaces del artefacto para descubrir defectos e identificar fallas.
	Prueba Estructural (caja blanca): Realizar pruebas de cobertura de algunas métricas (rutas de ejecución) en la implementación del artefacto.
Descriptivos	Argumentación informada: usar información de la base de conocimientos (por ejemplo, investigación relevante) para construir una argumentación convincente de la utilidad del artefacto.
	Escenarios: construir escenarios detallados alrededor del artefacto para demostrar su utilidad.

Estrategias tales como las pruebas de caja negra y de caja blanca, podrían ser válidas para evaluar la robustez estructural del artefacto, pero debido a que el entrono de uso del artefacto se encuentra en un contexto de aprendizaje, de los métodos de prueba sugeridos por Hevner et.al (2004), se optó por hacer una evaluación por medios experimentales, aplicando las siguientes estrategias:

a) Pruebas de usabilidad, para evaluar y mejorar las características de la interfaz de usuario del artefacto.

b) Pruebas experimentales para medir el efecto en el aprendizaje que tiene el artefacto en la población objetivo.

El detalle de la aplicación de estas dos estrategias se muestra en las siguientes secciones.

4.2.1 Prueba de usabilidad.

Nielsen (1993) señala que la usabilidad no es una propiedad unidimensional de las interfaces de usuario, sino que tiene múltiples componentes, asociados por lo general a cinco grandes atributos.

- Facilidad de aprendizaje, de manera que el usuario pueda empezar a utilizar rápidamente el sistema de información y realizar trabajo en él.
- Eficiencia, en el sentido de que una vez que el usuario aprendió a usar el sistema, éste favorezca una alta productividad.
- Facilidad de recordar, para que el usuario casual que ha dejado de usarlo por un tiempo, al regresar a él no tenga que aprender nuevamente desde cero.
- Disminuir la tasa de errores, en cuanto a que el sistema fomente que el usuario comenta pocos errores al usarlo y cuando de hecho los cometa, pueda recuperarse fácilmente de ellos.
- Satisfacción, en el sentido de que el sistema debe ser agradable de usar en el sentido subjetivo.

Así mismo, el advierte que probar un sistema con usuarios reales es fundamental y es en muchos casos una estrategia irremplazable, debido a que provee información directa sobre los problemas concretos de usabilidad que enfrentan tales usuarios al utilizar una interfaz de usuario.

4.2.1.1 Prueba piloto.

Se propone que antes de llevar a cabo cualquier prueba de usabilidad, deben especificarse los objetivos de la misma para identificar que estrategias de prueba pueden ser las más efectivas. Para esto debe hacerse la distinción entre una evaluación formativa, y una evaluación acumulativa. La primera sirve para ayudar a mejorar la interface como parte de un proceso evolutivo de diseño. La segunda tiene como objetivo evaluar la calidad general de una interface para decidir, por ejemplo, entre dos alternativas de diseño, o como parte de un análisis competitivo.

Así, el objetivo principal de una evaluación formativa es identificar detalladamente cuales aspectos de la interfaz son buenos y malos y como es que su diseño puede ser mejorado.

En relación con la recomendación de Nielsen, el objetivo de la prueba de usabilidad aplicada al prototipo era identificar elementos de la interfaz que pudieran ser mejorados, por lo que puede entenderse que la prueba aplicada sería de tipo formativo.

Entonces, la estrategia de prueba consistió en diseñar cinco tareas (ver Tabla 15) que fuesen representativas de la funcionalidad básica del prototipo y que ayudaran a identificar los puntos débiles de la interfaz del mismo. Dado que en este punto el alcance de la funcionalidad del primer incremento no era significativa, el diseño de las tareas no tendría que ser muy extenso.

Tabla 15. Versión preliminar de lista de tareas para prueba de usabilidad.

Tarea 1	Abrir el protocolo del problema “obtener la suma de los pares y el promedio de los impares” elaborado por la maestra Lizbeth Muñoz
Tarea 2	Regresar a la lista de problemas, entrar al protocolo del problema de la serie $1^2 + 2^3 + 3^4 + \ldots + n^m$

(…Continuación). Tabla 15. Versión preliminar de lista de tareas para prueba de usabilidad

Tarea 3	Encontrar el paso donde se usa la función POW
Tarea 4	Recorrer el protocolo hasta el final
Tarea 5	Regresar a la página de inicio.

Antes de llevar a cabo una prueba formal de usabilidad, es recomendable realizar una prueba piloto con pocos sujetos participantes, con el objeto de verificar si las instrucciones de las tareas son comprensibles, para contar con una referencia aproximada del tiempo necesario en realizar cada tarea y en general, verificar la practicidad del plan de prueba.

Para tal efecto, se solicitó la ayuda de un experto en el uso de tecnologías de información, a quien se sometió a una sesión de protocolo de pensamiento en voz alta (Knox, Bailey, y Lynch 1989; Krahmer y Ummelen 2004; J. Nielsen, Clemmensen, y Carsten 2002) (ver Figura 18), encontrándose la necesidad de combinar las tareas cuatro y cinco en una sola, colocar la tarea dos como primer tarea, y cambiar la forma de la tarea tres. Se registró que le tomó a la participante 14 minutos y 54 segundos realizar las cinco tareas.

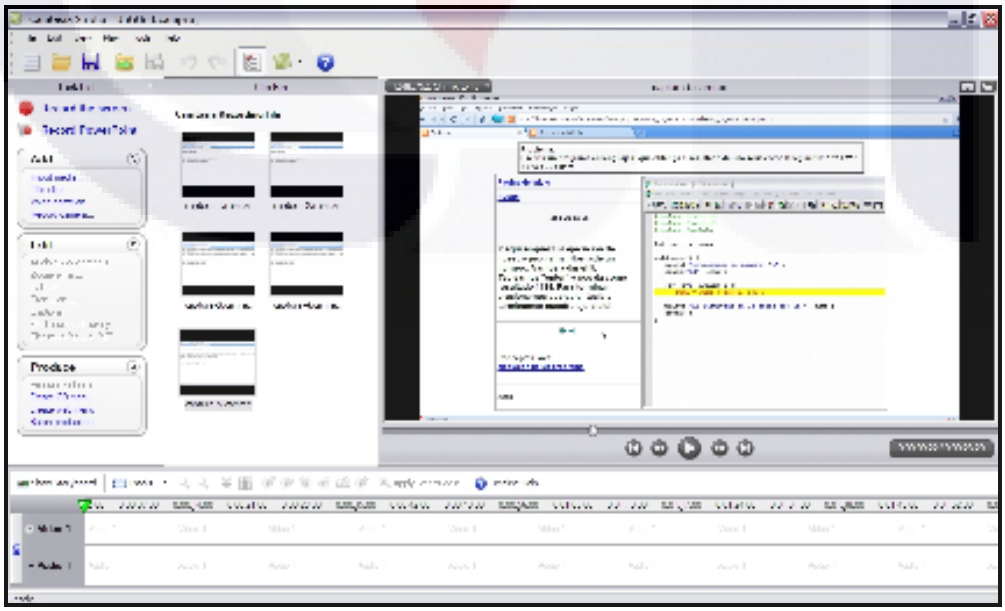


Figura 18. Ejemplo de prueba piloto mediante protocolo de pensamiento en voz alta.

4.2.1.2 Prueba formal.

Las tareas modificadas a raíz de la prueba piloto se muestran en la Tabla 16 y fueron aplicadas en la prueba de usabilidad formal bajo las siguientes condiciones:

Tabla 16. Tareas para prueba de usabilidad a partir de prueba piloto.

Tarea 1	Buscar y abrir el protocolo que corresponde al problema de obtener el resultado de la serie 1^2 + 2^3 + 3^4 +...+ n^m, resuelto por el maestro Francisco Javier López Rodríguez.
Tarea 2	Una vez dentro del protocolo, navegue hasta el paso 3 de 26 y lea el texto de cada paso.
Tarea 3	Moverse hasta el último paso del protocolo y visualizar **VARIAS VECES** el vídeo que muestra la ejecución y solución del problema.
Tarea 4	Vea la página de ayuda del visualizador de protocolos y después regrese a la página de inicio del sistema

a) Se diseñó un cuestionario en línea con 14 preguntas, para evaluar los temas “Interface” (5 preguntas), “Facilidad de uso” (3 preguntas), y “Facilidad de aprendizaje” (3 preguntas) (Ver Anexo II). Los reactivos eran preguntas cerradas de una opción en escala Likert de 1 a 7. Se incluyeron 3 preguntas para comentarios abiertos en cada uno de los 3 temas, para recolectar información cualitativa.
b) Se montó el prototipo Visor de Protocolos en un servidor Web de la Universidad Autónoma de Aguascalientes y se hizo disponible vía Internet[25].
c) Se solicitó a 17 estudiantes voluntarios de 8º semestre de la carrera de Licenciado en Tecnologías de Información de la Universidad Autónoma de Aguascalientes, que realizaran las tareas indicadas la Tabla 16, quienes las realizaron desde sus propias máquinas portátiles.
d) Después de realizadas las tareas, se aplicó un cuestionario (ver Anexo II) a los 17 participantes para buscar la retroalimentación de usuarios reales para la prueba formativa, según lo indicado por (Nielsen 1993, p.170).

[25] http://dsi.ccbas.uaa.mx/Protocolos

Los resultados de las preguntas cerradas se muestran en la Tabla 17.

Tabla 17. Resultados del cuestionario de la prueba de usabilidad.

Tema	Rango positivo			Rango Neutral	Rango negativo		
	Alto	Medio	Bajo		Bajo	Medio	Alto
Interface							
Encontrar los problemas	23.53%	58.82%	17.65%	0%	0%	0%	0%
Navegación entre los pasos	29.41%	35.29%	29.41%	0%	5.88 %	0%	0%
Visualizar el contenido de cada paso	35.29%	58.82%	0%	0%	5.88 %	0%	0%
Claridad del texto	64.71%	29.41%	0%	0%	0%	0%	0%
Combinación de colores	41.18%	35.29%	23.53%	0%	0%	0%	0%
Facilidad de uso							
Acceder al sistema	35.29%	52.94%	5.88%	5.88%	0%	0%	0%
Componentes requeridos	52.94%	41.18%	0%	0%	0%	0%	0%
Consistencia	41.18%	41.18%	17.65%	0%	0%	0%	0%
Facilidad de aprendizaje							
Cantidad de elementos en la interface	29.41%	52.94%	17.65%	0%	0%	0%	0%
Esfuerzo requerido	41.18%	41.18%	17.65%	0%	0%	0%	0%
Memoria de corto plazo	41.18%	41.18%	5.88%	0%	0%	0%	0%

De los datos resultados obtenidos de esta prueba, se observa que en general los usuarios no consideran que el prototipo tenga una mala usabilidad, ya que solo en dos de las once preguntas cerradas del cuestionario, indicaron una calificación negativa, y esto en un bajo porcentaje de los temas correspondientes (5.88%). Un dato interesante puede observarse en el reactivo de navegación, en donde el porcentaje de calificaciones (positivas) está divido (29.41%, 35.29% y 29.41%, respectivamente), lo que sugiere que a los usuarios les pudo parecer satisfactoria la navegación, pero que encontraron algunos problemas menores. Una situación similar se encuentra en el tema de combinación de colores y en menor medida el de encontrar los problemas. El resto de los temas puede concluirse que fueron satisfactorios para los usuarios.

La información cualitativa recolectada en las preguntas abiertas, permitió a los participantes escribir comentarios y recomendaciones sobre el prototipo. El resumen de los temas señalados con mayor frecuencia por los participantes puede verse en la Tabla 18. El detalle de todas las respuestas abiertas puede verse en el Anexo V.

Tabla 18. Retroalimentación cualitativa de la prueba de usabilidad.

Tema	Descripción	Numero de observaciones
Interface es sencilla	• Interface es sencilla, consistente, fácil de usar y recordar.	6
Mayor interactividad	• Mayor interactividad para no perder el interés. • Ir directamente a un paso específico del protocolo.	5
Incluir más elementos en pantalla	• Pantalla muy vacía. • Incluir más explicaciones. • Comentar el objetivo del problema y las estructuras básicas estudiadas en él.	4
Mejorar búsqueda de problemas	• No tener que leer texto de todos los problemas para seleccionar un protocolo	3
Mejorar la ayuda	• Ayuda no tiene botones de regresar a página anterior o al inicio	3
Botones de navegación	• Aumentar el tamaño de los botones de navegación.	3
Evitar pasos sencillos	• Condensar pasos muy sencillos (tal como declarar librerías al inicio)	1

Los datos cuantitativos y cualitativos recolectados por medio del cuestionario proporcionaron útiles guías que señalaron los puntos positivos y negativos de la interface, que era el objetivo de la estrategia de prueba. Ésta información se tomó en cuenta para hacer ajustes en la funcionalidad y diseño de la interface y producir un segundo incremento del prototipo.

Las modificaciones realizadas fueron las siguientes:

a) Agregar una funcionalidad de “ir al paso deseado” (ver Figura 19) en el visualizador de protocolos, para atender las recomendaciones cualitativas de

incrementar la interactividad (ver Tabla 18) y las respuestas cuantitativas del cuestionario en el tema de navegación. Esta opción permitía al usuario omitir algunos pasos considerados mecánicos y saltar directamente a un paso de interés en subsecuentes sesiones de estudio.

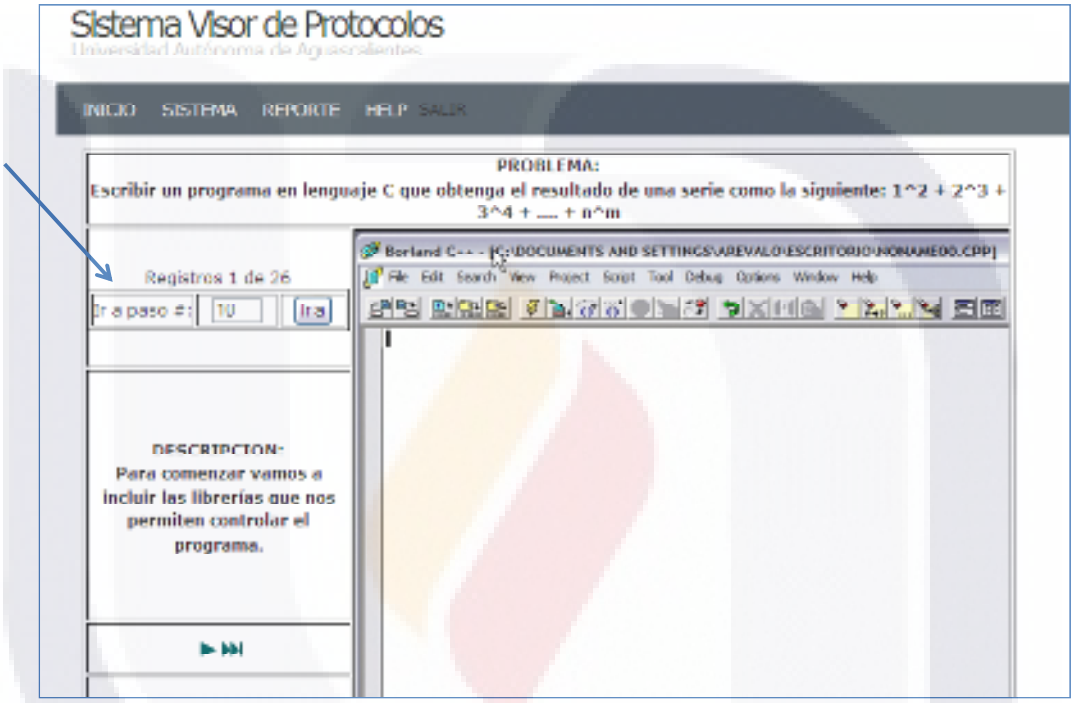


Figura 19. Opción de ir a un paso específico del protocolo verbal.

Es importante mencionar que hasta este punto en el desarrollo del prototipo, la funcionalidad indicada por el ISDT asociada a los “problemas por completar” (ver hipótesis de producto de diseño PROD5 y elemento de Meta-diseño MD9) no fue implementada en esta iteración, debido a restricciones de tiempo. Sin embargo, el diseño del modelo de datos asociado al prototipo (ver Figura 14) contempla y permite esta posibilidad. El tratamiento de esta situación se comenta en las recomendaciones para trabajos futuros (ver sección 6.5).

b) Agregar una opción que identificara y explicara en que paso del protocolo se encontraba el "punto focal" (Rist 1995, p.537) y que indicase al estudiante la operación principal del programa (ver Figura 20). Se consideró que esta opción atendía las observaciones de los usuarios de "incluir más elementos y más explicaciones en la pantalla". Una posibilidad interesante para futuras iteraciones, relacionada con esta petición, es el poder agregar una descripción general de la estrategia seguida en el protocolo, a manera de resumen.

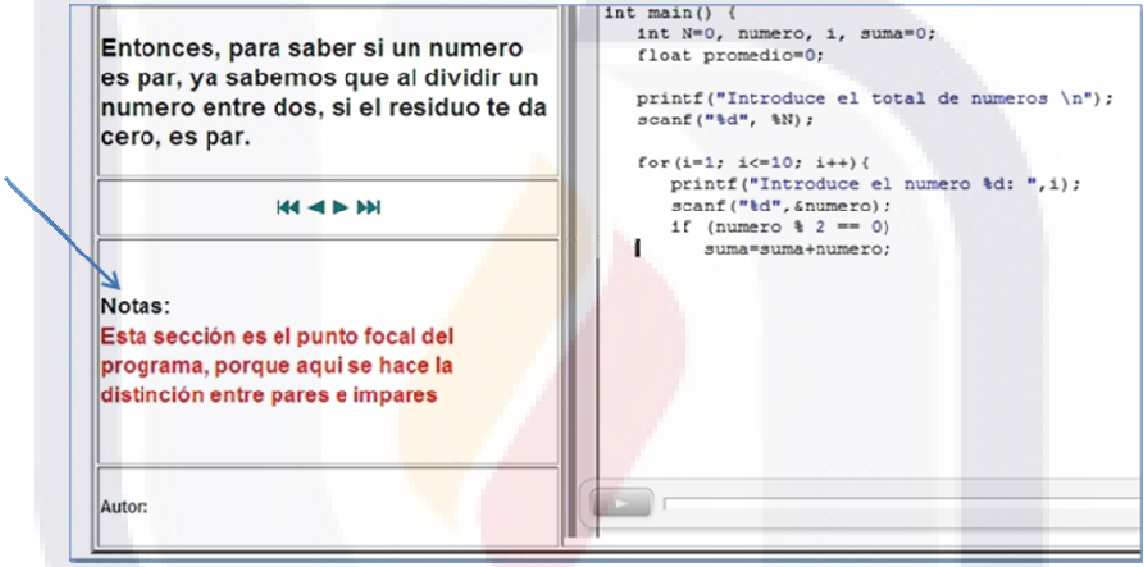


Figura 20. Identificar el punto focal de un programa en el prototipo visor de protocolos.

c) Incluir al inicio del sistema una opción de búsqueda por palabras clave del problema, categoría del problema o autor del protocolo. Dado que tres observaciones cualitativas relacionadas con este tema y los datos cuantitativos del cuestionario (Tabla 18, tema "mejorar búsqueda de problemas") recibió una calificación positivamente dividida, se consideró necesario incluir esta funcionalidad en la siguiente iteración del prototipo, por lo que también fue desarrollada (ver Figura 21).

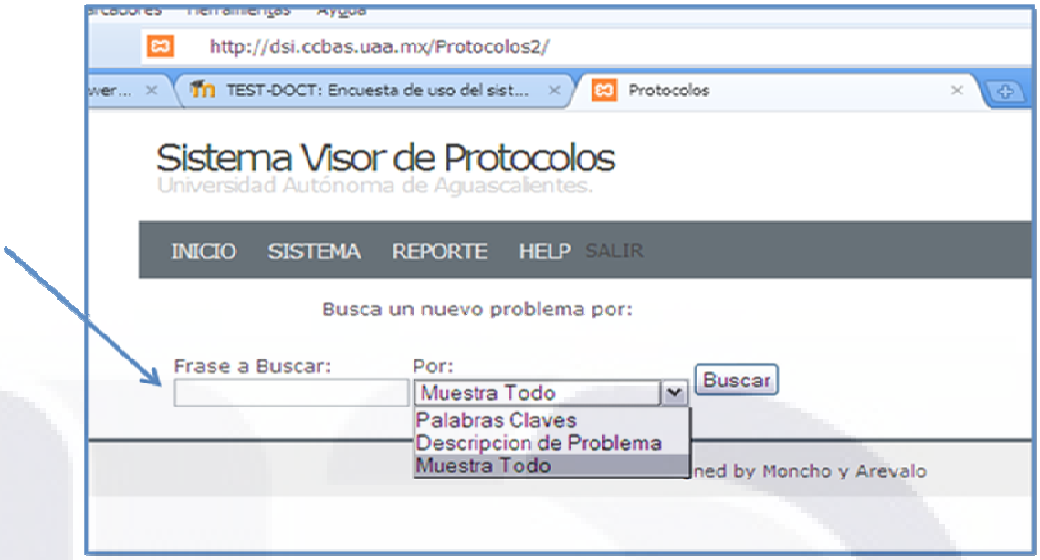


Figura 21. Opción de búsqueda específica de protocolos

Con base en lo descrito hasta este punto del presente capítulo, las hipótesis de producto de diseño (ver Tabla 9) PROD1 a PROD3 quedan comprobadas mediante el ejercicio de instanciación. La hipótesis de producto de diseño PROD4 (asociada a la factibilidad de incluir características de tipo "problemas por completar") queda sin verificarse, pero se argumenta que su implementación es factible dada su inclusión en el modelo de datos instanciado. Posibles estrategias para esta instanciación se discuten la sección 6.5. Debe recordarse también que el primer incremento del prototipo incluye solamente uno de los dos subsistemas descritos en la sección 4.1.1.

La hipótesis de proceso de diseño PROC1 (ver Tabla 11), asociada a la factibilidad de desarrollar Sistemas Visores de Protocolos Verbales mediante la estrategia de desarrollo de prototipos evolutivos, queda comprobada al aplicar dicha estrategia en el desarrollo del prototipo.

La hipótesis de producto de diseño PROD5, (asociada al aprendizaje significativo y transferencia de estrategias de programación, derivada a su vez del uso de un artefacto diseñado de acuerdo a las hipótesis PROD1 a PROD4),

tratará de comprobarse en la siguiente sección y de demostrarse en el capítulo cinco.

4.2.2 Modelo de investigación.

Para llevar a cabo la comprobación de la hipótesis de producto de diseño PROD 5 (ver Tabla 9), relacionada con la capacidad del artefacto para fomentar una transferencia de habilidades de solución de nuevos problemas básicos de programación, se propone el siguiente modelo de investigación (ver Figura 22), en donde el método de enseñanza es la variable independiente y se desea observar el efecto que éste tiene sobre el aprendizaje, medido en términos de transferencia de habilidades.

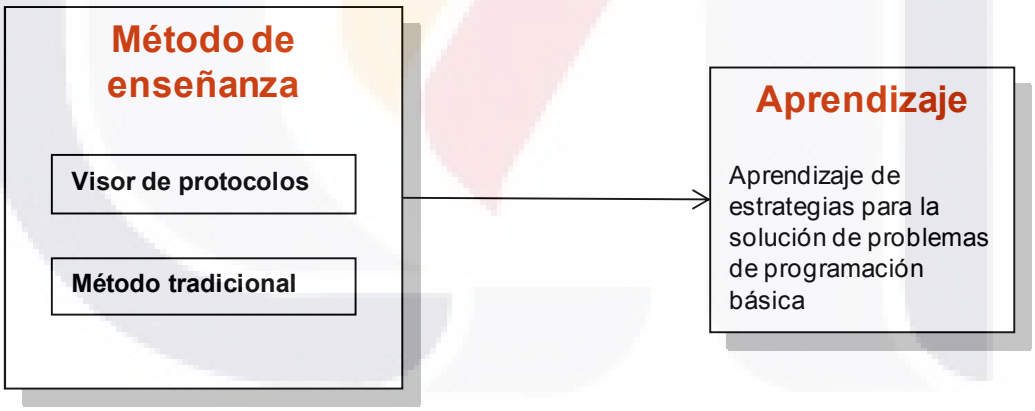


Figura 22. Modelo de investigación.

El significado de las variables del modelo de investigación propuesto se describe en la Tabla 19. Las condiciones de medición y manipulación de estas variables se comentan en la sección 4.2.3

Tabla 19. Descripción de variables de modelo de investigación.

Variable	Significado
Método de enseñanza	Variable independiente
Método tradicional	Se considera como método de enseñanza tradicional, aquel en donde participa un instructor humano, se utilizan materiales y estrategias didácticas tales como uso del pizarrón y solución de ejercicios en laboratorios de cómputo. Se argumenta que en los métodos tradicionales, no suele enfatizarse el conocimiento estratégico para escribir programas (Linn y Clancy 1992; Linn 1985; McGill y Volet 1997; Volet y Lund 1994).
Uso de visor de protocolos	El uso del artefacto visor de protocolos, con las características descritas en la Tabla 8 (Meta-diseño) y desarrollado mediante el proceso de diseño indicado en la Tabla 11.
Aprendizaje	Variable dependiente
	Medido en términos de la capacidad de un estudiante para aplicar habilidades que impliquen el uso estrategias de programación en la escritura de programas de computadora. Se maneja el concepto de transferencia (Mayer y Wittrock 1996) en el sentido de la capacidad de una persona de aplicar lo aprendido para resolver problemas nuevos. Las características del instrumento de medición de aprendizaje se describen en la sección 4.2.3.1.

4.2.3 Prueba experimental.

Una vez desarrollado y probado, se estuvo en condiciones de probar el artefacto instanciado a partir de la ISDT y así poner a prueba la hipótesis comprobable de producto de diseño PROD5 “Un SVP diseñado de acuerdo a las hipótesis PROD1 a PROD4 fomentará una transferencia de habilidades de

aplicación de estrategias en estudiantes de programación que lo utilicen en sus primeras etapas de aprendizaje" (ver Tabla 9).

Como ya se ha mencionado, un indicador de aprendizaje significativo tiene lugar cuando el estudiante es capaz de resolver nuevos problemas a partir de las habilidades adquiridas. Es decir, cuando tiene lugar una transferencia de conocimiento (Mayer y Wittrock 1996). Teniendo esto en mente, se consideró la conveniencia de evaluar el aprendizaje mediante un instrumento de medición en forma de examen escrito.

4.2.3.1 Instrumento de medición.

Para medir la capacidad del artefacto para fomentar una transferencia de habilidades y un aprendizaje significativo, se diseñó un instrumento consistente de un examen con tres problemas de programación básica a ser escritos a papel (ver Anexo III), para evaluar a su vez la capacidad de un estudiante de resolver problemas de programación básica que implicasen la habilidad sintáctica y la aplicación de estrategias para:

a) Reconocer problemas relacionados con repetición, selección y decisión.
b) Aplicar estructuras de repetición y decisión.
c) Reconocer y aplicar acumuladores y contadores.
d) Realizar cálculos relacionados con potencias.

El instrumento fue diseñado a partir de datos de exámenes previamente aplicados, tomados del banco de reactivos de la Academia de Programación de la carrera de Ingeniería en Sistemas Computacionales (ISC) de la Universidad Autónoma de Aguascalientes (UAA), específicamente de la materia de Lógica de Programación. La población de alumnos que resolvieron esos exámenes correspondió a estudiantes de primer semestre de ISC.

Se seleccionó aleatoriamente un grupo de 15 alumnos de ISC y los reactivos se seleccionaron de dos exámenes de Lógica de Programación (Primer Parcial y Segundo Parcial) de los cuales se tomaron los tres ejercicios del instrumento. Se asignó el mismo porcentaje (33.3%) a cada uno de los tres ejercicios del instrumento.

La escala de calificación fue estandarizada de 0 a 10. La estadística descriptiva resultante del análisis de los datos históricos de los reactivos del instrumento se muestra en la Tabla 20. Se observa que la media tuvo un valor de 5.7, la mediana fue de 5.8, la desviación estándar 2.27 y la moda fue 8.8.

Tabla 20. Estadística descriptiva del instrumento de medición.

Estadística Descriptiva		Calificación obtenida
N	Válidos	15
	Media	5.7340
	Error estándar de la media	.58806
	Mediana	5.8300
	Moda	8.83
	Desviación estándar	2.27753
	Varianza	5.187
	Asimetría	-.679
	Error estándar de asimetría	.580
	Curtosis	.879
	Error estándar de curtosis	1.121
	Rango	8.50
	Mínimo	.33
	Máximo	8.83

Una distribución de frecuencias es perfectamente simétrica cuando su mediana, moda y media aritmética coinciden. En este caso, la distribución no es simétrica, dado que la moda corresponde a 8.83, la media a 5.7 y la mediana a 5.83, pero estas dos últimas están muy cercanas. El coeficiente de asimetría (-.679) indica que la distribución es ligeramente asimétrica a la izquierda. Sin embargo, por la inspección visual del histograma de frecuencias (Figura 23), a pesar de no ser una distribución simétrica, se observa que la muestra sigue un

comportamiento normal, por lo que puede asumirse que los resultados arrojados por el uso del instrumento de medición no están sesgados y por lo tanto éste es confiable.

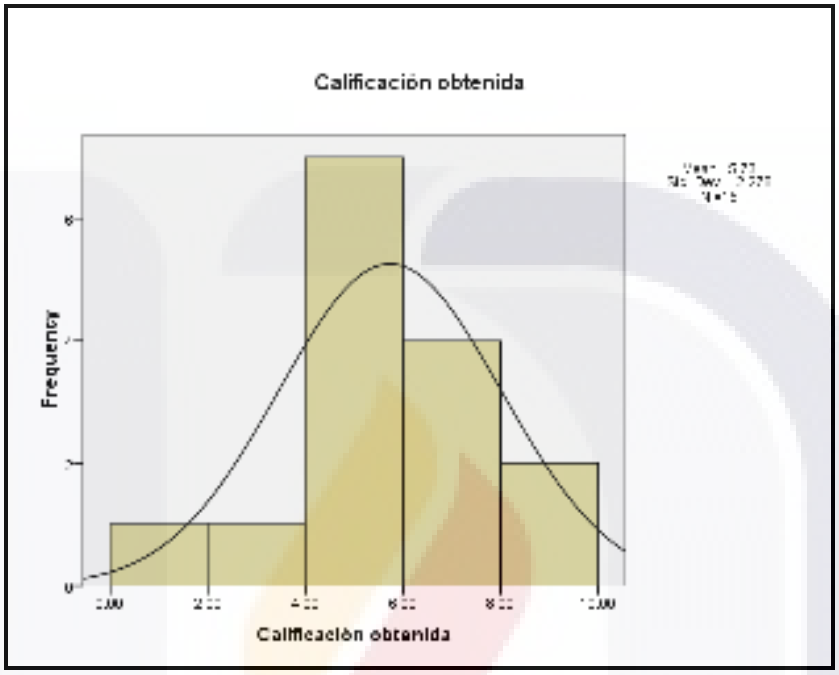


Figura 23. Histograma de frecuencias de la prueba piloto.

4.2.3.2 Condiciones del estudio experimental.

Para realizar la prueba experimental se formaron dos grupos de estudiantes de programación básica, matriculados en las carreras de Ingeniería en Electrónica (IE) y Sistemas Computacionales (ISC), respectivamente[26]. Ambos grupos tenían instructoras diferentes.

El grupo de estudiantes de programación de segundo semestre de Ingeniería Electrónica estaba formado por 20 participantes. Este grupo se utilizó como grupo de control (TRAD). Al momento de realizado el estudio, estos

[26] Semestre Enero – Junio 2010

alumnos habían estudiado los temas correspondientes al avance programático de la materia de Programación I, que comprendía el estudio de estructuras de datos en lenguaje C (vectores, matrices y apuntadores). El 55% de los alumnos de este grupo habían llevado materias de programación en sus respectivos bachilleratos de procedencia. El 5% de los participantes de este grupo eran mujeres y el restante 95% eran hombres. Para este grupo, los datos se recolectaron en las siguientes condiciones:

- Previo a la aplicación del instrumento de medición, a la instructora se le proporcionó el texto de cuatro problemas de programación básica, y se le solicitó los entregara a sus alumnos para que trataran de resolverlos por su cuenta, a manera de tarea.
- Los problemas entregados a la instructora y al grupo de control que correspondían a los mismos problemas cargados en el prototipo del Sistema Visor de Protocolos (ver Anexo IV), para buscar igualdad de condiciones en ambos grupos. También se buscó que los problemas del instrumento de medición fuesen semejantes a los cargados en el Sistema Visor de Protocolos, en cuanto a tener estrategias de solución similares.
- Un día después la maestra explicó la solución de los ejercicios a sus alumnos siguiendo métodos tradicionales (exposiciones verbales, uso del pizarrón y notas en cuaderno por parte de los alumnos). La explicación de la solución a los problemas le tomó a la instructora una sesión de clase (aproximadamente 50 minutos).
- Posteriormente se aplicó el instrumento de medición a los participantes de este grupo. El tiempo otorgado para resolver los problemas fue de una hora. No fue posible registrar los tiempos individuales de solución del examen de cada alumno.
- Como elemento de motivación, se ofreció a los participantes puntos extras para su calificación final. Esto entendiendo que la participación en el estudio les implicaba un esfuerzo adicional a su carga normal de estudios.

El grupo de estudiantes de programación de segundo semestre de ISC estuvo formado por 18 participantes. Este grupo se utilizó como grupo experimental (EXP). Al momento de aplicar el estudio, su avance temático correspondía al estudio de archivos en lenguaje C. El 41% de este grupo contaba con experiencia previa en materias de programación de sus respectivos bachilleratos de procedencia. El 6% de los participantes de este grupo eran mujeres y el 94% eran hombres. Para este grupo, las condiciones de recolección de datos fueron las siguientes:

- Los alumnos fueron llevados a un laboratorio de cómputo con computadoras con acceso a Internet, para poder acceder al prototipo. La configuración de las computadoras del laboratorio era la misma para todos los participantes, para controlar el posible efecto de una diferencia en el uso de los recursos tecnológicos.
- Se dio una explicación breve de 15 minutos acerca de las características de acceso a los problemas y la forma de navegación en los protocolos.
- Se pidió a los participantes que estudiasen detalladamente durante el resto de la sesión (45 minutos) todos los pasos de los cuatro protocolos cargados en el prototipo.
- La instructora de este grupo no participó en la explicación de los protocolos cargados en el sistema.
- Los problemas resueltos cargados en el prototipo (ver Anexo IV), correspondieron a los mismos ejercicios entregados a la instructora del grupo de control.

Posterior al estudio de los protocolos en el sistema, se solicitó a los participantes que contestaran el instrumento de medición. Se les dio una hora para contestar los ejercicios. No fue posible registrar los tiempos individuales de solución del examen de cada alumno de este grupo.

El análisis de los datos experimentales resultantes se describe en el siguiente capítulo, con el objetivo de comprobar o refutar la hipótesis asociada con la capacidad de un artefacto visor de protocolos de fomentar un aprendizaje significativo en estudiantes de programación (PROD5).

5 RESULTADOS.

No importa lo bella que sea tu teoría. No importa lo listo que seas. Si no va de acuerdo con el experimento, está equivocada.

-Richard Feynman, Físico teórico norteamericano (1918-1988).

Es una noción conocida entre profesores de programación que la calificación de exámenes presenta cierta dificultad debido a los múltiples criterios de evaluación que éstos pueden aplicar[27]. Es decir, algunos instructores pueden optar por un criterio de tipo "todo o nada", en donde si el programa (escrito en la computadora) se ejecuta sin errores y proporciona las salidas solicitadas, se califica con el valor más alto de una escala. Otros, pueden optar por dar un peso mayor o menor a ciertos tipos de errores de sintaxis. Otros pueden obviarlos. El proceso de calificación resulta aún más difícil cuando el instrumento de medición consta de programas escritos en papel.

En vista de lo anterior, buscando minimizar sesgos y para estandarizar en la medida de lo posible los resultados del instrumento, se plantearon los criterios generales de calificación que se muestran en la Tabla 21.

Tabla 21. Criterios de calificación del instrumento de medición.

Criterio	Descripción
Escala	Se usa una escala de 0 a 10.
Estrategia general	Se refiere a que si el programa escrito muestra indicios de que el alumno escribió en el programa una estrategia general considerada correcta, su calificación será por lo menos aprobatoria, con valor mayor o igual a 6. De lo contrario, si la estrategia general es incorrecta, su calificación máxima es de 5.
Errores de cálculo	Los errores relacionados con escritura incorrecta de fórmulas para realizar cálculos restan de 1 a 2 puntos.

[27] Esta noción es, por supuesto, anecdótica.

(...Continuación). Tabla 21. Criterios de calificación del instrumento de medición

Criterio	Descripción
Errores de lógica	Los errores de lógica (tales como construir incorrectamente una expresión en un ciclo o una decisión) restan de 1 a 2 puntos.
Errores de sintaxis significativos	Tales como la escritura incorrecta de un operador o de una instrucción. Se restan 0.5 puntos.
Omisiones sobre la petición de salida	Es decir, cuando el programa escrito no toma en cuenta las salidas indicadas en el problema. Se restan 2 puntos por cada omisión.
Ejercicio no resuelto	Se califica con 0.
Errores de sintaxis triviales	Tales como omitir un punto y coma en lenguaje C. No restan puntos

Se entiende que la validez académica de estos criterios puede ser debatible. Para el caso de la prueba experimental, se argumenta que el objetivo fue contar con un criterio de calificación uniforme y buscar consistencia en cuanto a otorgar un mayor peso a la demostración de la habilidad de aplicar estrategias correctas de solución de problemas de programación.

5.1 Análisis de varianza (ANOVA)

Para la prueba experimental (cuyas condiciones se describen en la sección 4.2.2.2) se recolectaron 38 observaciones correspondientes a 20 participantes del grupo de control (TRAD) y 18 del grupo experimental (EXP). Para analizar si existieron diferencias significativas en el aprendizaje de ambos grupos, se efectuó una prueba ANOVA (ver Tabla 22), observándose como resultado una diferencia estadísticamente significativa entre ambos grupos ($p<=.006$). En consecuencia se infiere que una de las dos estrategias instruccionales produce un mejor desempeño en los participantes.

Tabla 22. Resultados prueba ANOVA entre grupo experimental y de control.

	Suma de cuadrados	df	Cuadrado de medias	F	Sig.
Entre grupos	61.549	1	61.549	8.596	.006
Dentro de grupos	257.781	36	7.161		
Total	319.330	37			

5.2 Estadística descriptiva.

El análisis de la estadística descriptiva de los grupos (ver Tabla 23) puede indicar cuál de éstos tuvo un mejor desempeño en los temas estudiados. Así, puede observarse que la media del grupo EXP, tiene un valor de 6.33 y la del grupo de control uno de 3.79. Este primer dato proporciona indicios de un mejor rendimiento del grupo experimental, aunque la media de éste apenas sobrepasa el valor aprobatorio de 6. La mediana del grupo EXP corresponde a 7.5 y la del grupo de control a 3.7, lo que también puede sugerir una diferencia significativa en el desempeño de ambos grupos.

El coeficiente de asimetría del grupo EXP tiene un valor negativo (-.499), lo que indica que sus resultados se agrupan hacia los valores altos de la escala (asimétricos hacia la izquierda). Por el contrario, la asimetría del grupo TRAD es positiva (.689) lo que indica que sus datos se agrupan hacia la parte baja de la escala (asimétricos a la derecha). La desviación estándar del grupo EXP tiene un valor de 2.99 lo que sugiere que los datos de la muestra están muy dispersos respecto a la media. La desviación estándar del grupo TRAD es menor (2.35), lo que sugiere una menor dispersión de datos alrededor de su media. Los resultados anteriores se traducen en que en promedio, el grupo EXP tuvo un desempeño 25% mejor respecto al grupo TRAD, en relación a la escala de 0 a 10.

Tabla 23. Estadística descriptiva de prueba experimental.

		Calificación Final grupo EXP	Calificación Final grupo TRAD
N	Válidos	18	20
	Faltantes	2	0
Media		6.3389	3.7900
Mediana		7.5000	3.7000
Moda		3.30(a)	2.30
Desviación Estándar		2.99257	2.35683
Varianza		8.955	5.555
Asimetría		-.499	.689
Error estándar de asimetría		.536	.512
Curtosis		-1.412	.443
Error estándar de curtosis		1.038	.992

a Existen múltiples modas. Se muestra el valor más bajo.

5.3 Distribución de frecuencias.

La tabla de frecuencias (Tabla 24) del grupo experimental (EXP) muestra que solo el 38.9% de los participantes tuvo una calificación reprobatoria, que corresponde a 7 de los 18 participantes de este grupo. De los participantes con calificación aprobatoria, 7 obtuvieron un resultado entre la escala de 8.3 y 10 y cuatro participantes estuvieron entre el rango de 6.7 y 7.8. Esta distribución puede explicar la dispersión de datos indicada por el valor de la desviación estándar de la muestra (2.99), observándose que los resultados en general no están muy agrupados alrededor de la media.

Tabla 24. Tabla de frecuencias de calificaciones. Grupo experimental.

		Frecuencia	Porcentaje	Porcentaje Válido	Porcentaje Acumulado
Válido	1	1	5.6	5.6	5.6
	2.2	1	5.6	5.6	11.1
	2.7	1	5.6	5.6	16.7
	2.8	1	5.6	5.6	22.2
	3.3	2	11.1	11.1	33.3
	5.3	1	5.6	5.6	38.9

(…Continuación). Tabla 24. Tabla de frecuencias de calificaciones. Grupo experimental.

	6.7	1	5.6	5.6	44.4
	7.3	1	5.6	5.6	50.0
	7.7	1	5.6	5.6	55.6
	7.8	1	5.6	5.6	61.1
	8.3	1	5.6	5.6	66.7
	9	2	11.1	11.1	77.8
	9.2	1	5.6	5.6	83.3
	9.3	1	5.6	5.6	88.9
	9.5	1	5.6	5.6	94.4
	9.7	1	5.6	5.6	100.0
	Total	18	100.0	100.0	

La Tabla 25 muestra la distribución de frecuencias del grupo de control (TRAD), en donde se observa que el 85% de los resultados se encuentran en el rango entre 0 y 5.7 (17 observaciones) y las restantes 3 observaciones (2 con 6.7 y 1 con 9.7) corresponden al restante 15%. Como se indicó en el análisis de la estadística descriptiva, la desviación estándar (2.35) de las calificaciones obtenidas por este grupo indica que se encuentran más cercanas a su media, lo que sugiere un desempeño uniforme, pero pobre respecto al grupo experimental.

Tabla 25. Tabla de frecuencias de calificaciones. Grupo de control.

		Frecuencia	Porcentaje	Porcentaje Válido	Porcentaje Acumulado
Valido	0.7	2	10.0	10.0	10.0
	1	1	5.0	5.0	15.0
	1.3	1	5.0	5.0	20.0
	1.7	1	5.0	5.0	25.0
	2.3	3	15.0	15.0	40.0
	3.3	1	5.0	5.0	45.0
	3.7	2	10.0	10.0	55.0
	4.3	1	5.0	5.0	60.0

(…Continuación). Tabla 25. Tabla de frecuencias de calificaciones. Grupo de control

	4.7	2	10.0	10.0	70.0
	5	1	5.0	5.0	75.0
	5.3	1	5.0	5.0	80.0
	5.7	1	5.0	5.0	85.0
	6.7	2	10.0	10.0	95.0
	9.7	1	5.0	5.0	100.0
	Total	20	100.0	100.0	

Comparando la curva asociada a los histogramas de los dos grupos (ver Figuras 24a y 24b), se observa una forma platicúrtica en el grupo experimental, debido a que los resultados –como ya se ha mencionado– de este grupo se encuentran dispersos, ya sea entre el rango de calificación entre 0 y 4, o entre el rango de 5.3 y 10, estando sin embargo la mayoría de las calificaciones en este segundo grupo. En lo que respecta a la asimetría, en el grupo (EXP), se observa un sesgo hacia los valores altos de la escala. La curva asociada al histograma del grupo de control (TRAD) muestra una forma leptocúrtica, indicando que sus valores están más agrupados hacia los valores centrales de la muestra, la asimetría de este grupo muestra un sesgo en este caso hacia los valores bajos de la escala.

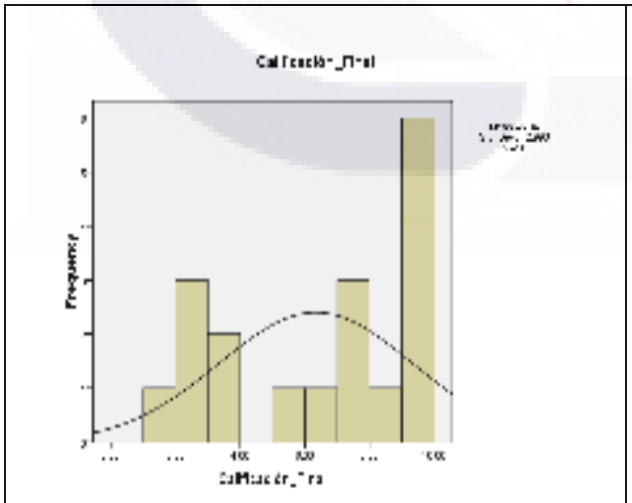


Figura 24. Histograma grupo EXP.

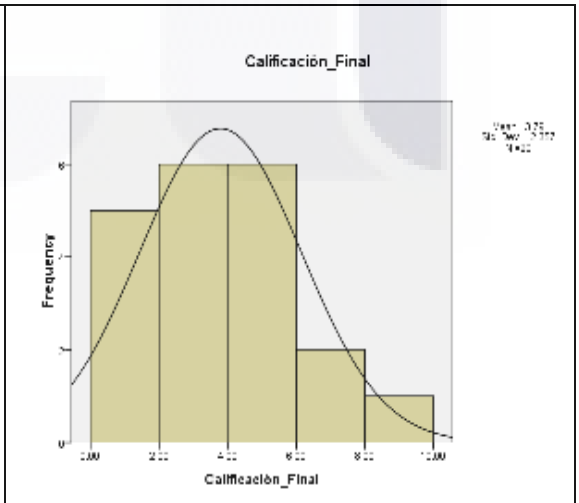


Figura 25. Histograma grupo TRAD.

5.4 Análisis de correlación

Dado que en la literatura (Fauxx 2006; Hagan y Markham 2000) se reporta un efecto positivo entre la experiencia previa y el rendimiento en el primer curso de programación, se optó por verificar si podía existir este tipo de relación causal en el aprendizaje de la programación de los participantes del estudio. En el grupo EXP el porcentaje de alumnos que habían cursado materias de programación en sus bachilleratos de procedencia correspondió al 41% y en el grupo TRAD el porcentaje fue de 55%.

Así, se calculó para ambos grupos el coeficiente de correlación de Pearson con las variables Experiencia Previa y Calificación (ver Tabla 26 y Tabla 27). Los valores de la variable Experiencia Previa se obtuvieron recolectando información de los propios participantes al momento del estudio experimental, solicitándoles indicaran su bachillerato de procedencia y si habían cursado materias de programación. Los valores de la variable calificación corresponden a los resultados obtenidos por la aplicación del instrumento de medición.

Tabla 26. Correlación entre experiencia previa y aprendizaje de la programación. Grupo EXP.

		Experiencia previa	Calificación
Experiencia Previa	Correlación de Pearson	1.000	.334
	Sig. (Bilateral)		.175
	N	18	18
Calificación	Correlación de Pearson	.334	1.000
	Sig. (Bilateral)	.175	
	N	18	18

El resultado del grupo EXP (Tabla 26) señala que no existe correlación significativa (0.175) entre la experiencia previa de este grupo y el rendimiento mostrado en la prueba. Por lo que se puede afirmar que para el experimento

aplicado no existe un efecto de la experiencia previa en el desempeño de los participantes de este grupo.

Tabla 27. Correlación entre experiencia previa y aprendizaje de la programación. Grupo TRAD.

		Experiencia	Calificación
Experiencia Previa	Correlación de Pearson	1.000	-.139
	Sig. (Bilateral)		.558
	N	20	20
Calificación	Correlación de Pearson	-.139	1.000
	Sig. (Bilateral)	.558	
	N	20	20

De igual manera, los resultados del grupo TRAD (Tabla 27) muestran que tampoco existe correlación significativa (0.558) entre la experiencia previa y el rendimiento de los participantes en la prueba.

5.5 Resultados finales

Dados los resultados de la prueba ANOVA, los valores de la estadística descriptiva, y el análisis de correlación de ambos grupos, puede concluirse que el grupo experimental (EXP) tuvo un mejor rendimiento que el grupo de control (TRAD), lo que sugiere a su vez un efecto positivo del artefacto visor de protocolos en el aprendizaje de los alumnos y una transferencia de habilidades. No se observó un efecto entre la experiencia previa de programación y la calificación obtenida, en ninguno de los dos grupos.

Es así que el análisis de los resultados mostrados en el presente capítulo permite comprobar la hipótesis de producto de diseño PROD5 (ver Tabla 9), asociada al fomento de transferencia de habilidades de aplicación de estrategias de programación, derivada del uso del artefacto. Sin embargo, dadas las condiciones del experimento, en donde la selección de los participantes no fue aleatoria y por lo tanto pudo no ser representativa, los resultados de este estudio no pueden extrapolarse como válidos para toda la población objetivo, que corresponde a todos los estudiantes de programación básica, de primer año de educación superior.

6 CONCLUSIONES.

¡Oh dicha de entender, mayor que la de imaginar o la de sentir!
-Jorge Luis Borges, escritor (1899-1986).

6.1 De los objetivos e hipótesis del estudio.

Habiendo planteado los componentes de una Teoría de Diseño de Sistemas de Información para Sistemas Visores de Protocolos Verbales (capítulo tres) y al haber pasado por el proceso de instanciación (construcción) de un artefacto derivado de tal teoría de diseño –que incluyó la realización tanto de pruebas de usabilidad, como la recolección de datos experimentales para medir su capacidad de apoyar el aprendizaje–, puede discutirse si los objetivos de la presente tesis fueron alcanzados o no, y si la hipótesis general del estudio puede ser corroborada con la evidencia empírica recolectada.

Respecto al Objetivo General:

"Aplicar la metodología ISDT para plantear una teoría de diseño de sistemas de información que proponga una clase de artefacto cuyas propiedades permitan la visualización de modelos mentales válidos, la visualización de estrategias de solución y el razonamiento metacognitivo de expertos al resolver problemas de programación, con la finalidad de fomentar un aprendizaje significativo y una transferencia de estrategias de solución de problemas en alumnos universitarios que estudian cursos introductorios de programación".

Se argumenta que:

El objetivo general fue alcanzado debido a que se logró aplicar la metodología ISDT, proponiendo una Teoría de Diseño de Sistemas de Información para Sistemas Visores de Protocolos Verbales (capítulo tres) que contiene las propiedades indicadas en su Meta-diseño (ver Tabla 8) y cuyo proceso de instanciación produjo un artefacto que muestra evidencia de fomentar un aprendizaje significativo de estrategias de solución de problemas de programación en los alumnos que participaron en la prueba experimental (capítulo cinco), bajo las condiciones indicadas en la sección 4.2.3.2 y con las limitaciones descritas en la sección 6.3.

Respecto a los objetivos específicos.

En el **objetivo específico uno** se indicó que el estudio debería:

"Plantear una teoría de diseño de sistemas de información mediante la metodología ISDT, que proponga una clase de artefactos de software para fomentar un aprendizaje significativo en la aplicación de estrategias de programación para la escritura de programas básicos".

Se argumenta que se alcanzó el objetivo específico uno, dado que se logró plantear una Teoría de Diseño de Sistemas de Información para Sistemas Visores de Protocolos Verbales aplicando la metodología ISDT. La teoría de diseño contó con todos los componentes indicados por la metodología: Meta-requerimientos (ver Tabla 7), Teorías núcleo de producto y de proceso (ver secciones 3.1.1. y 3.2.2), Meta-diseño (ver Tabla 8) e hipótesis comprobables de producto (Tabla 9) y de proceso de diseño (Tabla 11), según lo descrito en el capítulo tres de esta tesis.

En el **objetivo específico dos** se indicó que el estudio debería lograr:

"Diseñar y construir un artefacto de software derivado de la teoría de diseño planteada, que a su vez contenga propiedades sugeridas por otras teorías, tanto para el producto como para el proceso de diseño".

Se argumenta que se alcanzó este objetivo específico, al poderse construir un artefacto de software que tuviese como atributos de diseño aquellos indicados por el Meta-diseño de la ISDT para Sistemas Visores de Protocolos (ver Tabla 8) los cuales a su vez tomaron como referencia principios de la Teoría de Codificación Dual (ver sección 3.1.1.1.), el uso de la técnica de registro de protocolos verbales (ver sección 3.1.1.2), y el uso de ejemplos resueltos y problemas por completar (ver sección 3.1.1.3). El proceso de diseño tomó como teoría núcleo el modelo de prototipos evolutivos incrementales (ver 3.2.2.1).

El objetivo específico tres indicó que el estudio se propuso:

Probar empíricamente el artefacto construido bajo los principios de la teoría de diseño propuesta, tanto en sus propiedades de diseño, como en su efectividad para fomentar el aprendizaje.

Se alcanzó el objetivo especifico, ya que el artefacto fue sometido a pruebas de usabilidad (prueba piloto y prueba con usuarios reales) para depurar la interfaz de usuario del artefacto (ver sección 4.2.1) y una prueba experimental de efectividad para medir el aprendizaje (ver sección 4.2.3) de estudiantes de segundo semestre de las carreras de Ingeniería en Sistemas Computacionales y de Ingeniería en Electrónica que cursaban la materia de Programación I.

Respecto a la hipótesis general del estudio

"Un artefacto de software que se derive de una teoría de diseño de sistemas de información (ISDT) para clases de artefactos diseñados para fomentar el aprendizaje significativo de estrategias de solución de problemas de programación, *fomentará un desempeño en el aprendizaje significativamente mejor* en alumnos universitarios que cursen materias introductorias de programación, en contraste con aquellos alumnos de las mismas características que no lo utilicen".

> Se argumenta que la hipótesis **se comprueba** con la evidencia recolectada en el estudio. Se describieron las condiciones de la prueba experimental (ver sección 4.2.3.2) bajo el modelo de investigación descrito en la sección 4.2.2., en la cual se manipuló la variable "método de enseñanza", con dos grupos de participantes, utilizando uno de ellos el método de enseñanza tradicional (TRAD) y el otro (EXP) un artefacto Visor de Protocolos Verbales, registrándose un efecto positivo en la variable "Aprendizaje", encontrándose una diferencia estadísticamente significativa entre ambos grupos, dados los resultados dados por ANOVA ($p<=.006$) y mostrando el grupo EXP un rendimiento 25% mayor (en promedio) al grupo de control. Existieron limitantes y variables no controladas en las condiciones del estudio experimental que pudieron haber afectado los resultados. Estas limitaciones se discuten en la sección 6.3.

En cuanto a la **Proposición uno**, asociada a la hipótesis general, se argumenta que:

"El desempeño en el aprendizaje de un alumno puede medirse empíricamente en términos de su capacidad para escribir programas sintáctica y semánticamente correctos que solucionen problemas de programación básica".

Para medir el efecto en el aprendizaje se diseño y calibró un instrumento que constaba de tres ejercicios de programación básica y cuyo contenido se describe en la sección 4.2.3.1 y en el Anexo III. Al analizar calificaciones históricas obtenidas de estudiantes de programación de semestres anteriores, el instrumento mostró un comportamiento normal, permitiendo argumentar que tal instrumento es válido para medir la capacidad de un estudiante de escribir programas que requieran la habilidad de resolver problemas de programación básica que implicasen la habilidad sintáctica y la aplicación de estrategias para reconocer problemas relacionados con repetición, selección y decisión, aplicación de estructuras de repetición y decisión, reconocimiento y aplicación de acumuladores y contadores y realización de cálculos relacionados con potencias.

6.2 De la aplicación de la metodología.

El marco conceptual para la formulación de Teorías de Diseño de Sistemas de Información (ISDT) provee una referencia metodológica que combina el rigor científico de creación de teorías, aplicada al diseño, construcción y prueba de tipos especiales de artefactos de software para solucionar clases de problemas de difícil solución.

Metodológicamente es un marco de referencia útil, en tanto obliga al practicante a realizar una descripción detallada del problema de investigación a tratar. Esto permite enfocar la búsqueda de teorías, -de todas las ramas de la ciencia-, que describan principios para la construcción de artefactos de software; y dado que este marco se encuentra en la rama de la ciencia conocida como ciencias del diseño, su fortaleza radica en el principio de prueba por construcción. Es decir, la eficiencia de la teoría de diseño se mide en función del grado de efectividad del artefacto construido bajo sus propios principios.

En el curso de la presente investigación se encontró que la identificación de teorías núcleo fue la fase que presentó mayor dificultad. Esto debido a que en las primeras etapas, los llamados Meta-requerimientos no estaban lo suficientemente bien identificados. Esto llevó a estudiar y probar diversas teorías (ver por ejemplo ACT-R de Anderson, Bothell, y M. Byrne 2004; CFT de Spiro et al. 2003; Modelos instruccionales para resolución de problemas estructurados de Jonassen 1997; CLT de J. Sweller, van Merrienboer, y Paas 1998; y las Jerarquías de aprendizaje de Gagné 1968), que en última instancia debieron ser descartadas por no contar con elementos que pudiesen ser aplicados al problema. Una vez que los mencionados Meta-requerimientos fueron debidamente alineados con la evidencia mostrada en la literatura (en este caso, sobre el problema del aprendizaje de la programación), la búsqueda de teorías núcleo se simplificó significativamente.

6.2.1 De las hipótesis de producto y proceso de diseño.

Las hipótesis comprobables de producto de diseño (PROD1 a PROD3, ver Tabla 9) relacionadas con la factibilidad de construcción del artefacto descrito por el Meta-diseño (ver Tabla 8) fueron comprobadas durante el proceso de desarrollo del artefacto, descrito en el capítulo cuatro de esta tesis. La hipótesis PROD4, relacionada con la factibilidad de incluir problemas por completar en segmentos de un protocolo no fue comprobada, pero se argumenta que el modelo de datos diseñado permite su implementación (ver sección 6.5 en donde se describen posibilidades de construcción en este tema).

La prueba experimental del artefacto se asocia a la comprobación de la hipótesis PROD5 y los resultados descritos en el capítulo cinco, indican que tal hipótesis relacionada con la capacidad del artefacto de fomentar la transferencia

de estrategias de programación en alumnos principiantes de programación, quedó comprobada.

Por su naturaleza dual, una teoría de diseño describe las características de productos y también propone guías para el propio proceso de desarrollo. En este sentido, la hipótesis PROC1 relacionada con el proceso de diseño de artefactos visores de protocolos (ver Tabla 11), se comprobó mediante la construcción del artefacto usando el modelo de prototipos evolutivos incrementales, con énfasis en el diseño y prueba de usabilidad de la interfaz de usuario diseñada para fomentar la codificación dual de protocolos verbales. Los propios protocolos fueron registrados aplicando los principios propuestos por Ericsson y Simon (1993) y aumentados usando software de captura de video para registrar la actividad visual-mecánica de programadores expertos.

El traslado de los protocolos transcritos al artefacto, se realizó aplicando el método clásico de modelación Entidad-Relación (Chen 1976), (ver Figura 14). La implantación del prototipo se realizó bajo plataforma Web, que permitió la visualización eficaz de las características visuales del artefacto. En la Tabla 28 se muestra una síntesis de las hipótesis de producto y proceso de diseño del presente estudio junto con los resultados obtenidos.

Tabla 28. Síntesis de hipótesis de producto y proceso de diseño y resultados obtenidos.

Hipótesis de producto de diseño		**Resultado**
PROD1	Es factible diseñar un SVP que permita registrar, visualizar y editar protocolos verbales de programadores expertos, aplicados a la solución de problemas de programación.	**Comprobada**, mediante el registro de protocolos verbales de expertos usando software de captura de audio y video, la correspondiente transcripción de los protocolos resultantes (ver Tabla 12) y la posterior captura y edición en la base de datos del artefacto, derivada del modelo propuesto en la Figura 14.

(…continuación). Tabla 28. Síntesis de hipótesis de producto y de proceso de diseño y resultados obtenidos.

PROD2	Es factible diseñar un SVP que permita registrar distintas categorías de problemas de programación que se asocien a uno o a varios protocolos verbales.	**Comprobada**, por medio de la implementación del modelo de datos propuesto en la Figura 14, diseñado a su vez según los principios de teoría clásica de Chen (1976). Específicamente, el modelo de datos incluye una tabla con atributos para el registro de distintos tipos de problemas.
PROD3	Es factible diseñar un SVP que permita visualizar protocolos verbales aumentados con animaciones de código.	**Comprobada**. El modelo de datos mostrado en la Figura 14 incluye el atributo '*link_a_media_código*' en la tabla '*Verbalización*' cuyo valor apunta a un archivo de video que simula a su vez la escritura de código correspondiente a un segmento del protocolo verbal. El uso de una plataforma basada en Web permitió la visualización de este tipo de medios visuales (ver Figura 17).
PROD4	Es factible diseñar un SVP que permita complementar segmentos de protocolos verbales con preguntas por completar por los estudiantes.	**No comprobada**. No fue posible diseñar y desarrollar esta característica del artefacto, pero se argumenta que su implementación es posible si se utiliza el modelo de datos propuesto (ver tabla '*problemaPorCompletar*' del la Figura 14). En la sección 6.5 se proponen líneas de investigación y desarrollo para esta característica, asociadas a la combinación de Visores de Protocolos y Tutores Cognitivos. Esta característica resultó no ser crítica para la comprobación de la hipótesis PROD5.

(…continuación). Tabla 28. Síntesis de hipótesis de producto y de proceso de diseño y resultados obtenidos.

Hipótesis de producto de diseño		Resultado
PROD5	Un SVP diseñado de acuerdo a las hipótesis PROD1 a PROD4 fomentará una transferencia de habilidades de aplicación de estrategias de programación en estudiantes que lo utilicen en sus primeras etapas de aprendizaje	**Comprobada**, mediante la conducción de la prueba experimental (ver la sección 5.2.3 y el capítulo 6) que señaló un desempeño significativamente mejor (25%) en términos estadísticos (ver Tabla 23), del grupo experimental sobre el grupo de control. Este resultado sugiere que el artefacto es capaz de fomentar la transferencia de habilidades de aplicación de estrategias de resolución de problemas de programación mediante la visualización explícita de tales estrategias.
Hipótesis de proceso de diseño		**Resultado**
PROC1	Es factible diseñar un SVP por medio de la estrategia de prototipos evolutivos incrementales	**Comprobada**, mediante la construcción del artefacto por el método de diseño de prototipos evolutivos incrementales con énfasis en pruebas de usabilidad de la interfaz de usuario (ver sección 5.2.1).

6.3 Limitaciones del estudio.

En las condiciones del estudio (ver sección 5.2.3) se reporta que los grupos participantes tuvieron instructores diferentes. Esta variable (estilo de enseñanza) pudo tener influencia en el desempeño de los participantes de ambos grupos, aunque éste posible efecto no fue medido. Sin embargo, debe reportarse también que el contenido temático (es decir, al avance en el programa de la materia al momento del experimento) de ambos grupos fue el mismo, por lo que ambos

grupos tenían el mismo antecedente conceptual y de habilidades de programación.

Los resultados de la prueba experimental no pueden generalizarse como representativos de toda la población de estudiantes universitarios principiantes de programación, debido tanto al tamaño reducido de la muestra (EXP, n=18, TRAD, n=20), como a la posible no representatividad de la misma, ya que la selección de los participantes se realizó por disponibilidad y no por selección aleatoria.

Debido a la disponibilidad de los participantes en el estudio, el tiempo de exposición de los temas (es decir la explicación de los ejercicios con estrategias de solución similares a los del instrumento, tanto de manera tradicional como con el uso del artefacto) se redujo a dos sesiones de una hora por grupo. Este diseño experimental no permitió contar con una medición más detallada del efecto a largo plazo del artefacto en el aprendizaje de los alumnos.

Finalmente, algunas variables que pudiesen haber sido identificadas en investigaciones posteriores a la revisión de la literatura de esta tesis, pudieron haber introducido algún efecto no medido en el estudio experimental.

6.4 Aportaciones del estudio.

Un aspecto que debe considerarse al estimar las aportaciones del presente estudio, es el hecho de ser éste uno de los primeros que se realizan en México aplicando el marco ISDT, lo cual sienta un precedente para que otros investigadores consideren y apliquen esta forma de crear conocimiento, que busca un balance entre el rigor científico y la generación de artefactos de software relevantes.

A continuación se enumeran las aportaciones específicas del trabajo desarrollado en esta tesis, en los ámbitos de la Ingeniería de Software, el estudio y aprendizaje de la programación, la disminución de los índices de reprobación en estudiantes novatos en el contexto local y la posibilidad de más opciones de empleo en la industria local y nacional de software.

a) El planteamiento de una Teoría de Diseño para Sistemas Visores de Protocolos representa una contribución importante en el sentido de sentar las bases en el ámbito de la ingeniería de software para la creación de una nueva clase de artefactos que contribuirán a un mejor entendimiento y práctica del campo de la programación.

b) La Teoría de Diseño de Sistemas Visores de Protocolos puede ayudar a los instructores de programación básica en la construcción de artefactos (con la ayuda de ingenieros de software) que fomenten en sus alumnos un tránsito más eficiente por las primeras etapas de adquisición de habilidades de programación (ver en página 76, Anderson, Fincham, y Douglass (1997)) y hacia las últimas etapas de automatización de estrategias. Es decir, se pone a disposición de los profesores un nuevo tipo de herramienta que puede mejorar sustancialmente el desempeño de sus alumnos y con ello coadyuvar en la solución del fenómeno que originó el presente estudio.

c) Los antecedentes del problema se asocian a altos porcentajes de reprobación en materias introductorias de programación, tanto local (UAA 2007) como internacionalmente (T. Boyle 2003; Guzdial 2002; Dehnadi y Bornat 2006) Estos altos índices son reflejo de la dificultad real que supone la programación para los estudiantes que inician el estudio de esta disciplina (Bayman 1983; Dijkstra 1989; Arshad 2009; T. Boyle 2003; Jenkins 2002). El presente estudio podría ayudar a disminuir significativamente tales porcentajes de reprobación en alumnos novatos, mediante el uso continuo del artefacto visor de

protocolos durante las primeras etapas de su aprendizaje, en el contexto y condiciones descritos en la presente tesis.

d) Ayudar a los instructores de programación a describir explícitamente sus estrategias de solución de problemas y poder por lo tanto identificar distintos patrones, tipos y alcance de tales estrategias, de forma tal que se aporta nuevo conocimiento y evidencias al problema del aprendizaje de la programación. Instructores con mayor conocimiento del problema pueden a su vez ser otro medio para que los estudiantes sean directamente beneficiados y se mejore sustancialmente el índice de eficiencia terminal en las carreras que lleven materias de programación.

e) La aportación a la descripción del problema del aprendizaje de programación, en términos de factores críticos como los modelos mentales válidos, el conocimiento estratégico y la metacognición, puede ayudar a investigadores e instructores a desarrollar un vocabulario técnico que a su vez derive en métodos didácticos que enfaticen la importancia de la visualización explícita de tales factores.

f) La situación económica comentada en el capítulo uno (Litecky, Prabhakatar, y Arnett 2006; de Raadt, Watson, y Toleman 2003; Koong, Lai C. Liu, y Xia Liu 2002), sigue siendo vigente al momento de escribirse esta tesis, en donde la demanda por más y mejores programadores de computadoras es significativa, especialmente en el ámbito regional de Aguascalientes, debido a que con frecuencia se reciben peticiones de empleadores demandando este tipo de profesionales;. demanda que muchas veces es difícil satisfacer. Este orden de cosas brinda buenas oportunidades económicas para los estudiantes de Licenciaturas e Ingenierías afines a las Ciencias Computacionales que desarrollen la capacidad (y el gusto) por la práctica de esta fascinante disciplina que es la programación. La teoría de diseño planteada en esta tesis, puede contribuir a fomentar estas habilidades requeridas por la industria.

Se considera además que una aportación importante de esta tesis radica en el hecho de haber proporcionado evidencias tanto para investigadores como para profesionales del desarrollo de software, acerca del principio subyacente de que el desarrollo y uso de artefactos de software puede de hecho ayudar a mejorar el desempeño en el aprendizaje de los usuarios finales (en este caso estudiantes), no importando la complejidad del tema estudiado. Así, se abre la posibilidad de iniciar una nueva línea de investigación dentro del problema general del aprendizaje de la programación, que permitirá acumular conocimiento y experiencias para los practicantes del área.

6.5 Recomendaciones para trabajos futuros

Para próximos estudios que continúen las líneas de investigación sugeridas en la presente tesis, se recomienda diseñar y aplicar estudios longitudinales, que permitan medir el efecto que tendría el uso prolongado del tipo de artefacto aquí propuesto.

También se recomienda aumentar y diversificar el tamaño de la muestra, lo que permitiría –previo a un análisis estadístico–, generalizar los resultados obtenidos en la prueba experimental. Un estudio longitudinal requeriría también aumentar la cantidad de protocolos disponibles en el artefacto visor, lo cual representaría también una oportunidad de medir el efecto que tendría en el aprendizaje de los estudiantes, al contar con una mayor cantidad de estrategias de solución de problemas.

La funcionalidad del artefacto puede extenderse para mostrar protocolos sobre ejemplos resueltos en lenguajes de programación orientados a objetos, tales

como Java, C++ o C#, que también presenta dificultad a alumnos que inician el estudio de este paradigma de programación (Arévalo, Hernández, y J. Gómez 2007)

Una posibilidad interesante consiste en combinar el artefacto visor de protocolos, con otras de las tecnologías descritas en esta tesis, específicamente, los tutores cognitivos. Éste tipo de tecnología puede adaptarse para brindar la funcionalidad de los problemas por completar indicada en el Meta-diseño (ver Tabla 8, característica de Meta-diseño MD9).

Se han realizado estudios en esta línea (Arévalo y J. Gómez 2010) y se ha encontrado que existen herramientas de desarrollo rápido que facilitan su construcción (ver Figura 26). Las propiedades de retroalimentación dirigida de los tutores cognitivos pueden ayudar a los usuarios de los sistemas Visores de Protocolos, al ser adaptados a segmentos seleccionados de un protocolo verbal, haciendo las veces, como se ha dicho, de un problema por completar.

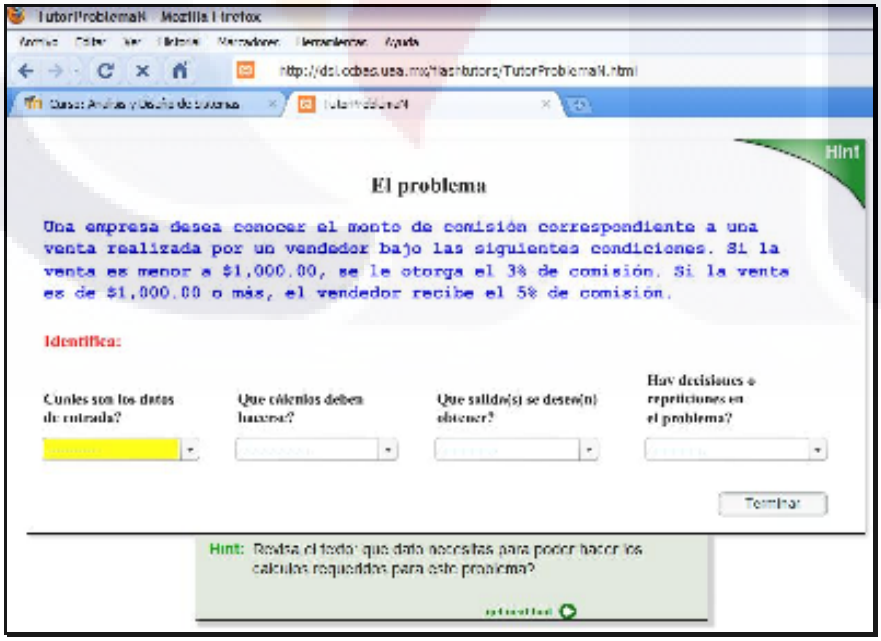


Figura 26. Ejemplo de tutor cognitivo.

Dada la naturaleza global del problema del aprendizaje de la programación, se recomienda también replicar el estudio en entornos socio-culturales distintos al aplicado en esta tesis, en donde la medición del comportamiento de estudiantes, instructores y contextos tecnológicos de diferentes países, puede ayudar tanto a medir la efectividad de la teoría de diseño, como a identificar otras variables que pudiesen también influir en el fenómeno.

ANEXO I. La naturaleza de las teorías de diseño

Walls et.al. (1992) señalan que la diferencia principal entre las teorías de las ciencias sociales y naturales y las teorías de diseño radica en cómo estas últimas tratan al comportamiento dirigido (es decir, los objetivos). Así, los objetivos no tienen sentido en las ciencias naturales (por ejemplo, la física), en donde se busca explicar el comportamiento de objetos y materia. En las ciencias sociales, si se puede tratar con objetivos. Por ejemplo, una teoría social puede tratar de explicar porque existen ciertos objetivos en una organización humana y/o predecir resultados asociados a objetivos. Pero el propósito de una teoría de diseño es promover el logro de objetivos.

Las siguientes ideas, tomadas de Walls, et. al (1992, pp. 40-41), ilustran la naturaleza de las teorías de diseño y sus diferencias respecto a las teorías de otras ramas de la ciencia.

1. ***Las teorías de diseño tratan a los objetivos como contingencias***. Mientras que los objetivos son extrínsecos a las teorías predictivas y explicativas, éstos son intrínsecos a las teorías de diseño. Un ejemplo de una ley explicativa puede ser “Y causa X”; la regla de diseño correspondiente puede ser “si se quiere alcanzar el objetivo X, entonces hacer que suceda Y”.

   Por ejemplo, F = MA es una ley de las ciencias naturales; la correspondiente teoría de diseño puede ser: “Si tu objetivo es acelerar la masa M = m, a una velocidad A = a, entonces provee la fuerza F = f

2. ***Una teoría de diseño nunca puede involucrar explicación o predicción puras***. Si ésta explica, explica las propiedades que debe tener un artefacto o como debe ser construido. Si predice, predice que un artefacto alcanzará sus objetivos hasta el punto en que éste posea los atributos descritos por la teoría, o hasta el punto en que los métodos descritos por la teoría sean usados para construir el artefacto.

3. ***Las teorías de diseño son prescriptivas***. Integran aspectos explicativos, predictivos y normativos hacia rutas de diseño de tipo "se puede" y "sucederá" que concretan diseños y usos más efectivos.

4. ***Las teorías de diseño son teorías "compuestas", que incluyen teorías de las ciencias naturales, ciencias sociales y matemáticas***. El plano prescriptivo provee el fundamento común para integrar estos tipos diferentes de teorías.

5. ***Las teorías explicativas dicen "lo que es", las teorías predictivas dicen "lo que será", las teorías normativas dicen "lo que debe ser" y las teorías de diseño dicen "como hacerlo / debido a".*** Aunque las teorías normativas también tienen que ver con objetivos, éstas son diferentes a las teorías de diseño. Las teorías normativas señalan que un agente *debería* buscar alcanzar un objetivo en particular (p.ej. una empresa debería maximizar sus ganancias), mientras que las teorías de diseño tratan sobre *cómo* alcanzar dicho objetivo.

6. ***Las teorías de diseño muestran como las teorías predictivas, explicativas o normativas pueden ponerse en práctica***. Si un artefacto que contiene las leyes de interacción de una teoría explicativa o predictiva es diseñado y construido y tal artefacto satisface sus requerimientos de diseño, entonces éste provee una medida de soporte empírico para la teoría

7. ***Las teorías de diseño son teorías de racionalidad procedural***. El objetivo de una teoría de diseño es prescribir tanto las propiedades que un artefacto debe tener si es que se desea alcanzar cierto objetivo, como los métodos de construcción de tal artefacto. Las propiedades del artefacto deben derivarse de la teoría de diseño. Las teorías de diseño involucran tanto la aplicación de teorías científicas, como el uso del método científico para probar tales teorías de diseño. Dado que los artefactos que resultan del proceso de diseño son construidos con elementos tomados del mundo natural o del social, están sujetos a las leyes que gobiernan dichos mundos. Por lo tanto, las teorías de diseño toman prestadas teorías naturales y sociales.

ANEXO II. Tareas y cuestionario para prueba de usabilidad.

La siguiente encuesta fue aplicada a estudiantes de 8º semestre de la carrera de Licenciado en Tecnologías de información. Posterior a la realización de las cuatro tareas indicadas abajo, se solicitó a los participantes (n=17) su retroalimentación acerca de aspectos de usabilidad e interfaz de usuario del prototipo. La encuesta fue llenada en línea.

Instrucciones:

Entrar al sistema visualizador de protocolos de problemas de programación básica que se encuentra en esta liga.

Una vez dentro del sistema, favor de realizar las siguientes tareas.

- **Tarea 1**: Buscar y abrir el protocolo que corresponde al problema de obtener el resultado de la serie 1^2 + 2^3 + 3^4 +...+ n^m, resuelto por el maestro Francisco Javier López Rodríguez.
- **Tarea 2**: Una vez dentro del protocolo, navegue hasta el paso 3 de 26 y lea el texto de cada paso.
- **Tarea 3**: Moverse hasta el último paso del protocolo y visualizar **VARIAS VECES** el vídeo que muestra la ejecución y solución del problema**.**
- **Tarea 4:** Vea la página de ayuda del visualizador de protocolos y después regrese a la página de inicio del sistema

Después de realizar estas cuatro tareas, favor de llenar la siguiente encuesta.

Muchas gracias por su valiosa aportación.

ENCUESTA DE USABILIDAD DEL PROTOTIPO DE SISTEMA VISOR DE PROTOCOLOS.

INTERFACE	
El sistema me permite encontrar con facilidad los problemas que me interesa estudiar	☐ No seleccionada ☐ Totalmente de acuerdo ☐ De acuerdo ☐ Algo de Acuerdo ☐ Neutral ☐ Algo en desacuerdo ☐ En desacuerdo ☐ Totalmente en desacuerdo
Cuando estoy viendo un protocolo, la interface me permite navegar adecuadamente por cada paso	☐ No seleccionada ☐ Totalmente de acuerdo ☐ De acuerdo ☐ Algo de Acuerdo ☐ Neutral ☐ Algo en desacuerdo ☐ En desacuerdo ☐ Totalmente en desacuerdo
No tengo problemas para visualizar los videos de cada paso de un protocolo	☐ No seleccionada ☐ Totalmente de acuerdo ☐ De acuerdo ☐ Algo de Acuerdo ☐ Neutral ☐ Algo en desacuerdo ☐ En desacuerdo ☐ Totalmente en desacuerdo

No tengo problemas para leer los textos debido a que el tipo de letra es adecuado	☉ No seleccionada ○ Totalmente de acuerdo ○ De acuerdo ○ Algo de Acuerdo ○ Neutral ○ Algo en desacuerdo ○ En desacuerdo ○ Totalmente en desacuerdo
La combinación de colores de la interface del sistema me parece adecuada	☉ No seleccionada ○ Totalmente de acuerdo ○ De acuerdo ○ Algo de Acuerdo ○ Neutral ○ Algo en desacuerdo ○ En desacuerdo ○ Totalmente en desacuerdo
Observaciones generales sobre la interface	
FACILIDAD DE USO	
La facilidad para acceder al sistema me parece adecuada	☉ No seleccionada ○ Totalmente de acuerdo ○ De acuerdo ○ Algo de Acuerdo ○ Neutral ○ Algo en desacuerdo ○ En desacuerdo ○ Totalmente en desacuerdo

Los componentes requeridos para poder ejecutar el sistema me parecen adecuados	◉ No seleccionada ○ Totalmente de acuerdo ○ De acuerdo ○ Algo de Acuerdo ○ Neutral ○ Algo en desacuerdo ○ En desacuerdo ○ Totalmente en desacuerdo
La consistencia que tiene el sistema para navegar de una página a otra me parece adecuada	◉ No seleccionada ○ Totalmente de acuerdo ○ De acuerdo ○ Algo de Acuerdo ○ Neutral ○ Algo de Desacuerdo ○ En desacuerdo ○ Totalmente en Desacuerdo
Observaciones generales sobre la facilidad de uso	

FÁCIL DE APRENDER

El número de elementos diferentes que estan en la página del visor de protocolos me parece adecuado	◉ No seleccionada ○ Totalmente de acuerdo ○ De acuerdo ○ Algo de Acuerdo ○ Neutral ○ Algo en desacuerdo ○ En desacuerdo ○ Totalmente en desacuerdo

El esfuerzo requerido para encontrar y visualizar un protocolo me parece adecuado	☉ No seleccionada ○ Totalmente de acuerdo ○ De acuerdo ○ Algo de Acuerdo ○ Neutral ○ Algo en desacuerdo ○ En desacuerdo ○ Totalmente en desacuerdo
Considero que es fácil recordar como acceder a las diferentes opciones del sistema	☉ No seleccionada ○ Totalmente de acuerdo ○ De acuerdo ○ Algo de Acuerdo ○ Neutral ○ Algo en desacuerdo ○ En desacuerdo ○ Totalmente en desacuerdo
Observaciones generales sobre la facilidad de aprender	

ANEXO III. Instrumento de medición

UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES
CENTRO DE CIENCIAS BÁSICAS.
DEPARTAMENTO DE SISTEMAS ELECTRÓNICOS/SISTEMAS DE INFORMACIÓN

Ejercicios de programación básica.

Aguascalientes, Ags. Abri-Mayo de 2010

Nombre:	
Carrera:	
Semestre:	
Grupo:	
Preparatoria de procedencia	
¿Llevó Ud. materias de programación en el bachillerato?	

Resuelva los siguientes problemas, escribiendo con claridad el código en lenguaje C.

1. Según el teorema de Pitágoras, el cuadrado de la hipotenusa es igual a la suma del cuadrado de los catetos (c2 = a2 + b2). Utilizando este concepto, es posible conocer de que tipo es un triángulo:

   Si $C^2 = a^2 + b^2$ entonces es un triangulo rectángulo
   Si $C^2 < a^2 + b^2$ entonces es un triangulo acutángulo
   Si $C^2 > a^2 + b^2$ entonces es un triangulo obtusángulo

   Escribir un programa que permita leer los dos catetos, calcule la hipotenusa y a partir de estos datos, imprima el tipo de triángulo que es.

2. En una cafetería se sirven solo dos tipos de platillos llamados: continental y mediterráneo, con los siguientes precios: 40 y 55 respectivamente. El programa se debe repetir mientras se desee capturar notas y para cada nota se debe solicitar la cantidad de platillos 1 consumidos así como la cantidad de platillos 2 consumidos. Al finalizar de capturar las notas se desea obtener la cantidad de platillos consumidos de cada tipo, los ingresos que generaron cada uno de ellos, el total de ingresos del restaurante y el nombre del platillo que más se consumió.

3. Escriba un programa que obtenga y muestre en pantalla el resultado de calcular la siguiente serie, en donde el valor de **n** deberá ser solicitado al usuario.

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ldots +- 1/n$$

¡Gracias por su amable participación!

ANEXO IV. Problemas cargados en el prototipo del Sistema Visor de Protocolos (SVP)

Al momento del estudio, los siguientes problemas estaban resueltos y disponibles para los alumnos del grupo experimental en el prototipo del Sistema Visor de Protocolos.

- Escribir un programa en Lenguaje C que al recibir como dato N números enteros obtenga la suma de los números pares y el promedio de los impares.
- Escribir un programa para calcular el factorial de un número entero.
- En una tienda de deportes se venden solo dos tipos de bicicletas llamadas 'crown' y 'space', con los siguientes precios: $20,000 y $30,000 respectivamente. El programa se debe ciclar mientras se desee capturar notas y para cada nota se debe solicitar tanto la cantidad de bicicletas crown, como la cantidad de bicicletas space compradas. Al terminar de capturar las notas, se desea obtener la cantidad de bicicletas vendidas de cada tipo, las ventas totales de cada tipo de bicicleta, el total de ventas por ambos tipos de bicis y el modelo de bici más vendido.
- Escribir un programa en lenguaje C que obtenga el resultado de una serie como la siguiente: $1^2 + 2^3 + 3^4 + \ldots + n^m$

ANEXO V. Retroalimentación cualitativa de prueba de usabilidad.

Respuestas abiertas sobre el tema de Interface

Interface	
Comentario	**Temática**
Los colores me parecen adecuados, aunque el acomodo de los marcos creo que confunde al momento de avanzar con los botones, y el tipo de letra no me parece muy amigable	Acomodo de los marcos confunde. Fuente inadecuada
En general me parece bien, la fuente es adecuada con un color que no lastima la vista	Fuente adecuada
Al consultar la ayuda del sistema y querer regresar a una etapa anterior o al inicio del sistema lo tuve que realizar por medio de los botones de Internet explorer, podría haber algún tipo de interfaz que me permita navegar desde la ventana de ayuda.	Fallas en la ayuda
Hacer un poco mas interactiva la interface por que a la larga vas perdiendo el interes y mas si ya estamos empapados del tema.	Incluir más interacción
La interface muy bien, porque ayuda al programador a resolver dudas ademas te indica paso a paso lo que debes hacer.	Interface apropiada
Creo que se podria aprovechar mas la parte de texto que esta a lado del sistema para explicar mas acerca del video	Más explicación en texto
Esta muy bien pero pienso que la interaccion estaria un poco mejor, puesto que si no se interactua mas, se puede poner un poco aburrido.	Incluir más interacción
Me parece que los botones de navegacion deberian ser un poco mas grandes.	Botones de navegación mas grandes
Hace falta la opcion de ir especificamente a x paso ya que si por alguna razon se equivoca el usuario y avanza al final tendra que recorrer otras vez los pasos.	Ir directamente a un paso del protocolo
Me parece quen los botones de navegacion deberian ser un poco mas visibles, y si se pudiera tener al final del tutorial una especie de repaso donde se pueda interactuar.	Botones de navegación mas grandes
Me gustaria que hubiera un botón para reproduccion continua, que pueda ser como un play y un pause para no tener que dar tantos clicks al terminar un video y seguir con el otro. Si pudiera cambiar un poco el diseño de colores ya que se ve un poco burdo el html plano, pero de ahi en fuera el acomodo y la facilidad de uso es buena. Para no saturar el servidor tambien seria de alguna utilidad tener el video completo en un sevidor de video.	Incluir un modo de reproducción contínua. Incluir colores más llamativos
La pantalla principal es adecuada, pero la de los pasos (videos) es demasiado escueta.	Interface de los pasos demasiado sencilla
En lo personal me parece una interface sencilla pero directa, ya que explica todo detalladamente	Interface sencilla apropiada
Solamente quiero agregar que al momento de presentar un problema estaria mejor si la parte escencial del programa como las librerias y el main la pusieran en un solo paso pero claro la explicacion que dan esta muy bien	Condensar pasos iniciales sencillos

Facilidad de uso	
Comentario	**Temática**
Muy facil de usar para cualquier usuario. solo me parece que los botones de avanzar estan demasiado pequeños	Botones navegación mas grandes
Algo que me parecio bien de la presentacion es que va lento dejando oportunidad para escribir y analizar lo que se escribe.	Videos adecuados
Es accesible, simple de manejar, muy logico, aunque estaria bien que algunos objetos se mostraran mas vistosos	Objetos mas llamativos
Ninguna	
Siento que se ve muy vacia la pantalla que puede tener mas interactividad	Mas interactividad
Si, aunque creo que algunos objetos pueden mejorarse para facilitar aun mas la navegacion.	Objetos para facilitar navegación
esta un poko lento pero kreo k es por la RIUUA.....	Lentitud de carga
faltan algunos links regresar al inicio o a la pagina anterior, ejemplo en el link ayuda	Links de ayuda
De la página de ayuda no es posible ir directamente a la página principal, debería existir un liga para ir directamente a ella sin tener que dar click en "Back"	Links de ayuda
No existe una dificultad para hacer uso de los protocolos, ya que cada paso esta explicado	Facil de usar

Facilidad de aprendizaje	
Comentario	**Temática**
Totalmente de acuerdo ya que me parece muy bueno el concepto, y me hubiera servido mucho hace años	Buen concepto
Ninguna	
Son buenas, ya que explican paso a paso, todo lo relacionado con la programacion desde un punto de vista como principiante	Buena idea explicar paso a paso
Es muy sencillo y faicl de usar para aprender.	Sencillo y fácil de aprender
Siento que como son pocos elementos los que se encuentran y es mas visual el ejercicio se tiene mayor facilidad de aprender	Facil de aprender por características visuales
Si se piensa que se puede hacer en un tipo de materia en la que se pueda adaptar la herramienta, seria excelente.	Sugerencia de adaptar herramienta a una materia de programación
Me parece muy bueno, ya que va paso por paso	
Seria bueno poner algunos tool tip en algunos componentes, así como en el video, ya que se torna dificil comprender que se quiere decir cuando en el video se subraya alguna palabra clave, aunque se describa en el texto.	Poner Tool Tips en componentes y en video
Me gustaria tambien que tuviera los objetivos (resumen) que se tratan en el tema, es decir por ejemplo este tiene: recursividad en c-ciclo for impresion en pantalla, elevar al cuadrado (pow) detener pantalla (getch) impresion de variables a pantalla etc.	Incluir los objetivos del problema resuelto
Para encontrar los protocolos es requerido leer el texto completo, sería más facilmente identificable si hubiera una columna que especificara el problema (sin ningún otro texto) resuelto y luego ya la columna de la descripción completa.	Búsqueda de protocolos dificil
Por lo mismo que es sencillo pero directo, se puede aprender con facilidad ya que hace hace enfasis en los temas importantes haciendo que el alumno pueda recordar con mas facilidad los elementos asi como su sintaxis de las diferentes situaciones. Felicidades por la realizacion de este sistema	Facil de aprender
ME PARECE QUE ES UNA HERRAMIENTA MUY UTIL PARA LOS ALUMNOS ES ALGO FACIL DE USAR	Herramienta es útil y facil de usar

ANEXO VI. Publicaciones generadas.

The Effect of Learning Objects on a C++ Programming Lesson

Carlos Argelio Arévalo Mercado
Autonomous University of Aguascalientes
México
carevalo@correo.uaa.mx

Estela Lizbeth Muñoz Andrade
Autonomous University of Aguascalientes
México
elmunoz@correo.uaa.mx

Juan Manuel Gómez Reynoso
Autonomous University of Aguascalientes
México
jmgr@correo.uaa.mx

ABSTRACT

The subject of teaching computer programming has been widely studied for the last 25 years. Different approaches and disciplines have given important knowledge about the problem, mostly cognitive science and the use of software tools and technology to aid teaching. In recent years, Learning Object (LO) technology has received world wide attention within the distance education community. However, literature research shows little experimental and empirical evidence about the effectiveness of LOs in academic performance. In this exploratory study, the effect of using LOs in a computer programming lesson is reported. Descriptive statistics and T test results show that there is no significant difference in student's performance, under the conditions of the study.

INTRODUCTION

Learning to program is difficult for many students. Dropout and failure rates in introductory programming at undergraduate courses are evidence to the fact that learning to program is a difficult task (Wiedenbeck & Kain, 2004). It can be said that algorithmic thinking is complex: more advanced concepts are layered on top of others, learned previously (Jenkins, 2002; Machanick, 2005).

Even though the subject of learning to program has been widely studied for the last 25 years, within several disciplines and from different approaches (psychological/cognitive, sociological, and software tools, among others) there is not yet a definite solution and understanding to the problem.

In the line of software tools to aid learning, learning objects (LOs) are receiving a lot of world wide attention. LOs are generally understood as digital learning resources that can be shared and accessed via Internet and used in multiple contexts (Wiley, 2000). The basic premise of LOs is to offer scalable and individually adaptive instruction, which can even be generated on the fly

according to the learner needs by intelligent semantic technologies (Gibbons, Nelson & Richards, 2000). And so, the adoption of LO technology has seen a significant increment within the distance learning community. Important efforts are made to establish standards (ISO, 2003), quality measures (Archambault, 2003), and efficient repositories. Substantial amounts of human and financial resources are being channeled to LOs related projects (CISCO, 2000).

However, the LO approach is not without criticism. For example, Jaakkola (2004, p.4), states: *When considering the effect of learning objects on students' learning performance, it is important to understand that it is impossible, and irrelevant, to separate the learner and the learning content to be learned from the context in which learning occurs.*

More so, research literature focused on the pedagogical effectiveness of LOs is limited (Moisey, 2003). There is little empirical evidence of the effect of LOs on academic achievement of students (Jaakkola, 2004; Kujansuu, 2005).

The purpose of this exploratory study is to statistically measure the effects of LOs in a first year C++ programming lesson, and find if students exposed to this technology (in a limited time frame) show better academic performance.

METHODOLOGY

In this study, we will describe an exploratory study that consisted of comparing the performance of two first year C++ undergraduate programming groups of students, under two different conditions:

1) Using traditional, teacher-led instruction, and
2) Using learning objects.

The study was conducted in Autonomous University of Aguascalientes (UAA), México.

The content was focused on the subject of file handling in C++. This subject was selected because it coincided with both the lecture plan and the thematic progress of the course. Thirty five first year computer science students participated in the study. Two groups were created: one using traditional methods (TRADG), and the other using LOs (LOG).

Before the formation of the groups, students with previous programming experience (Byrne, 2001; Holden & Weeden, 2005; Fauss, 2006) were identified and later randomly distributed within the two groups.

The study was conducted in the second half of the semester, so it was assumed that the participants already knew the basic programming concepts and structures (sequence, decisions, loops, variables) of the C++ language.

Structure of the Teaching Sessions

Each group was given a two hour session with the following structure:

- A lecture about the subject of file handling in C++ (reading and writing).
- Free, individual study/use of corresponding learning materials; and
- Solution of a small test

To control the teaching style variable, the same teacher conducted both sessions.

Both sessions were held in the same laboratory, using a projector.

Description of the Test

To minimize a possible waste of time, the students were asked to answer a written exam. This was also designed to avoid the effect of stress and anxiety due to compilation errors.

The test consisted of two exercises: one to demonstrate knowledge of "write" C++ related instructions, and the other related to the "read" counterpart instructions. The established scoring criteria were the following:

- Include the correct libraries (iostream.h ; fstream.h)
- Include the correct instructions to open files (ofstream write/read). This had to be done in the correct context of the program.
- A logically correct use of a "for" loop.
- Include the correct writing and reading instructions (write; read.getline)
- Include the correct instructions to close files (write.close, read.close) in the correct context of the program

Each one of these requirements was assigned one point. The complete test summed up a total maximum score of 10 points.

Description Of The Learning Objects

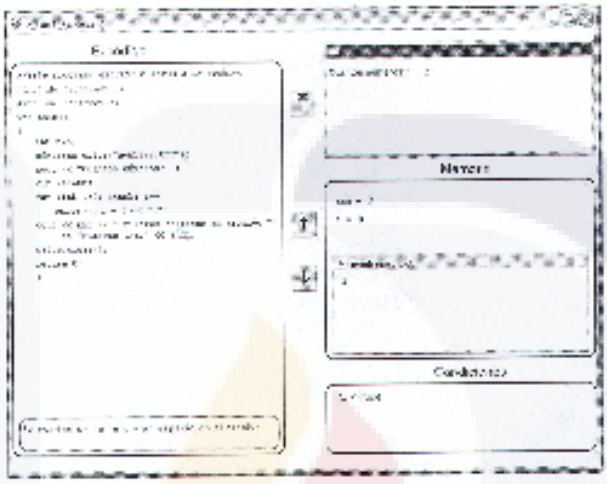
A more detailed description of the LOs used in the study is required. These LOs are part of the CodeWitz[1] international repository project, which is formed by several higher educations institutions. Its main objective is to help in the instruction of the basics of computer programming, specifically C++ and Java programming languages, through the use of high quality reusable learning objects.

For our study, the LOs were translated for a Spanish speaking audience.

[1] http://www.codewitz.net/

The LOs interface consisted primarily of clicking three navigation buttons (see Figure 1), to go forward and backwards through the execution of an example program (in this case, writing and reading files in C++).

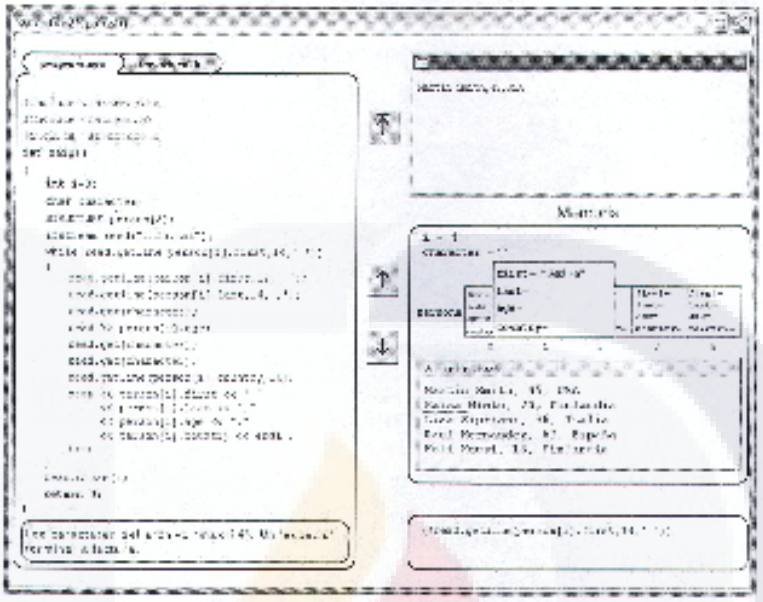
Figure 1: Adapted LO from the CodeWitz Project, for learning C++ file writing sentences.



The LOs have five different sections (see Figures 1 and 2):

- "The code", which shows the example program.
- An explanatory section of each one of the lines of the program (the content of these sections changes when the user navigates the program) located at the lower left of the LO.
- An "Output" section, to simulate how the computer screen could give input and output in the monitor.
- The "memory" section, to show the state and contents of variables and data structures. This section is particularly important, given that previous cognitive studies (Norman, 1983; Fixx, Wiedenbeck & Scholtz, 1993; Ramalingam & Widenbeck, 2004; Ma & Wood, 2007) have identified "proper mental models" as a relevant factor. The visualization style tries to follow a "debug like" behavior (Kajansun, 2006).
- Finally, a "Conditions" section to show the state of the logical conditions guiding the behavior of the loops and decision structures of the example program.

Figure 2: Adapted LO from the CodeWitz Project, for learning C++ file reading sentences.



As stated before, this particular design tries to reinforce the mental model of the student using visualization and showing what happens "inside" and "outside" of the program. This is also in accordance to the "notional machine" concept stated by DuBoulay (1989).

- ***Conditions for the Traditional Method Group (Tradg)***

For the TRADG group (n = 18), the teacher gave one lecture of the subject of writing and reading files in C++ using a PDF file, containing the theory and two examples. The approximate time was 15 minutes for theory and 10 minutes for explanation of the examples. After this, a 20 minute period was given for the students to freely study and experiment with the PDF examples. Past this time, the teacher asked the students to complete the written test (described previously). Individual completion times were recorded. A one hour limit was given to the participants.

Conditions for the Learning Objects Group (Log)

For the LOG group (n = 17), the lecture was made using the same PDF file given to the TRADG group, *minus the examples*, which were substituted with the LOs described above.

The teacher and the LOG group students used the LOs for approximately 15 minutes for the explanation of the examples. After that, the students used the objects freely for another 20 minutes. For the last part of the session the LOG group participants were asked to complete the same written test. A one hour limit was also given to the participants of this group.

Observation of the Participants

The teacher conducting the sessions was asked to take notes about the behavior of the participants, to provide complimentary qualitative data to the analysis. She recorded the following:

- The participants of the LOG group were found to be nervous and stressed about using the LOs. Comments such as "how do we use this?" were recorded.
- In general, the LOG group took more time to complete the test and 53% did not complete the second exercise of the test. In comparison, 35% of the TRADG group did not complete the second exercise of the test.
- The participants of the TRADG group were less anxious and nervous about the test.

RESULTS

Descriptive statistics obtained for both groups are shown in Table 1.

Table 1: Descriptive statistics of the study.

Group Statistics

	grupo	N	Mean	Std. Deviation	Std. Error Mean
Exercise A	1	18	3.1111	2.11824	.49927
	2	17	3.6176	2.10304	.51006
Exercise B	1	18	3.0000	2.25222	.53321
	2	17	2.2941	2.51263	.60945
Grade obtained in the exam	1	18	6.1111	4.23066	.99718
	2	17	5.9118	4.11262	.99746

An initial analysis of the data shows that the mean of the TRADG group was slightly superior (6.11) to that of the LOG group (5.91). This results are somehow discouraging and surprising. It is evident by the data obtained that the overall performance of the LOG group was lower than the TRADG.

Only in the first exercise (Exercise A) the LOG group (mean = 3.61) had slightly better performance than the TRADG group (mean = 3.11). However, for Exercise B, the performance of the LOG group was clearly inferior (mean = 2.29) than that of the TRADG group (3.00).

To corroborate what descriptive statistics seem to indicate, a standard t test was applied to both groups, with the following results:

Table 2: T test results for TRADG and LOG groups.

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means					95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
[illegible]	Equal variances assumed	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	Equal variances not assumed			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	Equal variances assumed	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	Equal variances not assumed			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	Equal variances assumed	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	Equal variances not assumed			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

As expected (see Table 2), the t test results are not significant in any case (Exercise A = .483, Exercise B = .388, Final Grade = .889). The null hypothesis for equality of means has to be accepted, meaning that there is no significant difference in the performance of both groups.

Given the results, it was considered that no further statistical analysis was needed.

DISCUSSION AND FUTURE STUDIES

Clearly, the results obtained were not the ones expected; however, they provide hints about what factors may have influenced this phenomenon. The qualitative report provided by the teacher that conducted the study, lead us to think that the participants of the LOG group likely felt pressured and anxious (Bresnan, 1998). Another factor could have been the limited time of exposure to the learning objects, even though the user interface seemed to be very easy to use. It is possible that the participants of the LOG group could have perceived that they had two different assignments: learn to use the LOs, and complete the test.

As mentioned previously, there are few studies that show empirical evidence about the effectiveness of LOs regarding student achievement (Moisey, 2003). The results of our study seem to concur with those found by Jaakkola (Jaakkola, 2004). In his study, three experimental studies using LOs were reported, and two of them did not show any significant differences in the performance of the participants (high school students). However, a third experiment did show significant differences due to the combination of simulation and hands-on experimentation (the topic covered by this experiment was learning of DC circuit design concepts).

In summary, the results of our exploratory study seem to indicate two main ideas:

a) A computer programming instructor cannot assume that the sole use of LOs, in a single lecture, will make a significant difference in the performance of students. There are other important contextual factors that may yet have to be identified to improve academic achievement. On the other hand, a more prolonged exposure to LOs has to be explored and measured under carefully controlled conditions.

b) In accordance to other experimental studies, the combined use of simulation and hands-on experimentation seems to be a promising line of research and development in the field of learning computer programming.

It can be suggested that the field of LOs can benefit with more experimental studies to give instructors a clearer vision of circumstances under which LOs can be more effective to academic performance.

Finally, in a purely speculative spirit, a kind of learning artifact that interactively gives both "guided experimentation" and visual feedback to the student, seems to be a desirable goal. However, the time and cost associated with this kind of development could be a significant barrier.

REFERENCES

Arabambault, J. V. J. C. N. K. B. A. (2003). Learning Object Evaluation: computer-mediated collaboration and inter-rater reliability. *International Journal of Computers and Applications*, 25(3).

Brosnan, M. J. (1998). The impact of computer anxiety and self-efficacy upon performance. *Journal of Computer Assisted Learning*, 14, 223-234.

Byrne, P. & Lyons, G. (2001). The effect of student attributes on success in programming. ACM SIGCSE Bulletin, 33(3), 49 - 52 .

CISCO (2000). Reusable learning object strategy. Definition, creation process, and guidelines for building. Version 3.1.

Du Boulay, J. B. H. (1989). Some difficulties of learning to program. Lawrence Erlbaum Associates, Hillsdale.

Fauxx, R. (2006). Impact of pre-programing course curriculum on learning the first programming course. *IEEE Transactions on education*, 49(1), 11-15.

Hixx, V., Wiedenbeck, S. & Scholtz, J. (1993). Mental representations of programs by novices and experts. Conference on Human Factors in Computing Systems Amsterdam, The Netherlands ACM Press New York, NY, USA .

Gibbons, A. S., Nelson, J. & Richards, R. (2000). The Nature and Origin of Instructional Objects. In The Instructional Use of Learning Objects. Bloomington: Association for Educational Communications and Technology.

Holden, E. & Weeden, E. (2005). Prior Experience and New IT Students. *Issues in Informing Science and Information Technology*, 2, 189.

ISO (2003). Business Plan for JTC1/SC36 (Standards for Information Technology for Learning, Education, and Training).

Jaakkola, T. N., S. (2004). Final Report on CELEBRATE Experimental studies. Learning Objects - A lot of smoke, but there is fire? Turku, University of Turku, Finland Educational Technology Unit.

Jenkins, T. (2002). On the difficulty of learning to program. 3rd annual LTSN-ICS Conference. Loughborough University, LTSN Centre of information and computer sciences.

Kujansuu, E. (2006). Using program visualization learning objects with non-major students with different study background. Proceedings of Methods, Materials and Tools for Programming Education conference, Tampere. Finland.

Ma, L. F. J. R., M & Wood, M. (2007). Investigating the viability of mental models held by novice programmers. ACM SIGCSE Bulletin, 39(1), 499-503.

Machanick, P. (2005). Peer Assessment for action learning of data structures and algorithms. Australasian Computing Education Conference. New Castle, Australia.

Moisey, S. M., A. (2003). Fulfilling the promise of learning objects. In Handbook of Distance Education, Lawrence Erlbaum. 1st ed., 323.

Norman, D. A. (1983). Some observations on mental models. Erlbaum, Hillsdale, NJ.

Ramalingam, V. L., D.& Wiedenbeck, S. (2004). Self-Efficacy and mental models in learning to program. Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education. Leeds, United Kingdom ACM Press New York, NY, USA.

Wiedenbeck, S. L., D.& Kain, V.N.R. (2004). Factors affecting course outcomes in introductory programming. 16th Workshop of the psychology of programming interest group. Institute of Technology, Carlow, Ireland.

Wiley, D. (2000). The Instructional Use of Learning Objects. Bloomington, IN. Association for Educational Communications and Technology.

UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

INVESTIGACIÓN Y CIENCIA
DE LA UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

Oficio: INV- 366/09

M. EN C. CARLOS ARGELIO ARÉVALO MERCADO
CENTRO DE CIENCIAS BÁSICAS
DEPTO. DE SISTEMAS DE INFORMACIÓN
P R E S E N T E

La informo que el artículo titulado "**Un estudio piloto sobre el efecto de los tutores cognitivos para la enseñanza de conceptos básicos de programación**" a cargo de Usted y del Dr. Juan Manuel Gómez Reynoso, fue aceptado para su publicación en el Núm. 46 de la Revista "Investigación y Ciencia de la Universidad Autónoma de Aguascalientes", periodo Enero-Abril 2010.

Sin otro particular por el momento, reciba un cordial saludo.

A T E N T A M E N T E
"SE LUMEN PROFERRE"
Aguascalientes, Ags., 7 de Septiembre de 2009

LIC. ROSA DEL CARMEN ZAPATA
EDITORA DE LA REVISTA INVESTIGACIÓN Y CIENCIA
DE LA UNIVERSIDAD AUTONOMA DE AGUASCALIENTES
ISSN 16654412
Certificado de Licitud de Título 12284
Certificado de Licitud de Contenico 8497

kZ/mrg

Departamento de Apoyo a la Investigación. Edificio 1-B, Segundo Piso
Av. Universidad No. 940, Cd. Universitaria, C.P. 20130, Aguascalientes, Ags.
Teléfonos (449) 9 10-74 42 y 43, Fax (449) 9 10-74 43
Contacto:
revistaiyc@correo.uaa.mx
http://www.uaa.mx/investigacion/revista

UN ESTUDIO PILOTO SOBRE EL EFECTO DE LOS TUTORES COGNITIVOS PARA LA ENSEÑANZA DE CONCEPTOS BÁSICOS DE PROGRAMACIÓN.

M. en C. Carlos Argelio Arévalo Mercado[28]
Juan Manuel Gómez Reynoso, PhD[29].

RESUMEN

La enseñanza de la programación presenta problemas recurrentes a alumnos de primer año de licenciatura, debido a la complejidad de su estructura de conocimiento. Diversos factores cognitivos han sido identificados en la literatura indicando un efecto en el rendimiento de los alumnos. Herramientas basadas en TI han sido desarrolladas para ayudar en el aprendizaje de la programación, contando con atributos de diseño y resultados diversos. El presente estudio reporta la aplicación de un estudio piloto con dos grupos estudiantes de nivel medio del estado de Aguascalientes, usando prototipos de Tutores Cognitivos. Se aplicó una evaluación para medir el grado de retención y se aplicaron pruebas estadísticas para analizar los datos. Los resultados preliminares no muestran diferencias estadísticamente significativas en el rendimiento de ambos grupos. Se argumenta que la ausencia de un instructor humano y la falta de una capacitación presencial previa, entre otras variables, pudo influir en los resultados obtenidos.

Palabras clave: Tutores Cognitivos, Programación, Diseño Instruccional, Teoría de Flexibilidad Cognitiva, Software para la enseñanza, E-learning

ABSTRACT

Learning to program is a recurring problem to first year undergraduate students, given its complex knowledge structure. Several cognitive factors have been identified in literature to have an effect in student performance. Tools based on Information Technology have been developed over time to aid apprentices in learning to program, with varying results and design attributes. This study reports the use of prototype cognitive tutors with two groups of high school students in the state of Aguascalientes, Mexico. An evaluation of performance was made and statistical tests were applied to collected data. Preliminary results show no statistical significant difference between the two groups. It is argued that the absence of a human instructor and the lack of face-to-face training could have influenced the results of the experimental group.

Key Words: Cognitive Tutors, Programming, Instructional Design, E-Learning, Cognitive Flexibility Theory, Software for Teaching.

[28] Profesor Investigador, Departamento de Sistemas de Información, Centro de Ciencias Básicas. UAA. Correo electrónico: carevalo@correo.uaa.mx
[29] Profesor Investigador, Departamento de Sistemas Electrónicos, Centro de Ciencias Básicas. UAA. Correo electrónico: jmgr@correo.uaa.mx

INTRODUCCIÓN

El problema de la enseñanza de la programación.

La enseñanza de la programación, entendida como la habilidad para escribir programas de computadora, es el tema de numerosas investigaciones a nivel mundial. La literatura reporta de manera recurrente (Dijkstra, 1989; Milne & Rowe, 2002) que los alumnos tienen dificultades para aprender a programar, lo cual se refleja en altos niveles de reprobación. Es claramente reconocido que la naturaleza del tema de la programación es compleja, debido, entre otros factores, a que su estructura tiende a ser jerárquica y no lineal, en tanto que los conceptos que la conforman están fuertemente relacionados y resulta poco efectivo enseñarlos de manera aislada y secuencial (Du Boulay, 1989; Jenkins, 2002; Milne & Rowe, 2002).

Dada la complejidad del problema, los investigadores han estudiado el tema desde diversas perspectivas y a la fecha han encontrado evidencia sobre algunos factores que influyen en la capacidad del estudiante para aprender a programar. Se habla de que la experiencia previa al primer curso de programación (Hagan & Markham, 2000; Holden & Weeden, 2005) afecta el desempeño del alumno durante los cursos introductorios. Los modelos mentales también influyen en el rendimiento del aprendiz de programador (Bayman, 1983; Fixx, 1993; Hegarty, 1993; Ma, 2007). Aspectos cognitivos como la llamada autoeficacia (Heggestad, 2005; Ramalingam, 2004; Wiedenbeck, LaBelle, & Kain, 2004), entendida como “lo bien que uno puede ejecutar cursos de acción requeridos para llevar a cabo situaciones prospectivas” (Bandura, 1982, pág. 122) y la ansiedad computacional (Brosnan, 1998) son factores sobre los cuales se tiene evidencia de un efecto en el rendimiento de los estudiantes. Finalmente, se habla de que la habilidad matemática (Hu, 2006; White, 2003) y los estilos de aprendizaje (Sadler-Smith & Smith, 2004; Thomas, Ratcliffe, Woodbury, & Jarman, 2002) también tienen influencia sobre la facilidad de aprendizaje del tema.

Como puede observarse, la gran cantidad de factores que afectan el rendimiento del estudiante de programación hacen que sea sumamente difícil, desde el punto de vista pedagógico, el diseño de estrategias de enseñanza efectivas que vayan más allá de los métodos tradicionales. Por otro lado, es reconocido que el modo de enseñanza por medio de tutores humanos –en donde un tutor humano experto enseña a uno o dos alumnos– produce un efecto significativamente mejor (Bloom, 1984) que la enseñanza convencional dentro de un salón de clase (donde un profesor enseña a un grupo aproximado de 30 alumnos). De tal suerte, que los métodos de enseñanza apoyados por TI buscan reproducir el efecto (du Boulay, 2000) que tiene esta modalidad de enseñanza personalizada.

Herramientas para la enseñanza de la programación basadas en Tecnologías de Información.

Los investigadores a lo largo del tiempo han desarrollado herramientas basadas en Tecnologías de Información (TI) buscando apoyar el proceso de enseñanza de la programación. La naturaleza de estas herramientas –y los resultados reportados– son sumamente diversos y una clasificación exhaustiva de ellos rebasa el alcance del presente artículo[30]. En general, las herramientas de apoyo a la enseñanza de la programación tienen en común los siguientes atributos.

- Ayudas visuales. Estas herramientas proporcionan apoyo al aprendiz, mediante animaciones del comportamiento de algoritmos (Hamer, 2004; Naps, 1998) o de conceptos complejos tales como la recursividad (Jehng, 1999).
- Multimedia. El uso de audio, video e interactividad también ha sido utilizado para ayudar en la comprensión de conceptos de programación (Chansilp, 2004; Cooper, 2003; McKeown, 2004).
- Minilenguajes. Este tipo de aplicaciones (Brusilovsky, 1998; McIver, 1999) buscan reducir la complejidad de los lenguajes de programación tradicionales, disminuyendo la cantidad de funciones disponibles y aumentando la usabilidad.
- Uso de inteligencia artificial. Los tutores inteligentes son una tecnología que intenta brindar retroalimentación dirigida al aprendiz de programador, al tiempo que rastrea patrones de uso (Brusilovsky, 1995; J. R. Anderson, Corbet, & K. R. Koedinger, 1995; Kumar, 2006).
- Reutilización. En este ámbito, los llamados objetos de aprendizaje (Wiley, 2000) prometen la reutilización de contenidos en diversos contextos por medio de estándares para su ensamble y búsqueda (Boyle, 2006; Kujansuu, 2006; Matthíasdóttir, 2006; Moisey, 2003; Neven, 2002).

En este contexto, los objetivos del presente estudio piloto consistieron en medir el efecto de un tipo especial de tutor inteligente sobre el aprendizaje de los principios básicos de la programación en alumnos de nivel preparatoria. El modelo de investigación aplicado considera al desempeño/aprendizaje de la programación como variable dependiente y al método de estudio como variable independiente.

Como objetivo secundario se buscó detectar cuales variables ambientales o cognitivas adicionales que no fueron controladas, pudieron haber tenido un efecto significativo en el proceso de transferencia de conocimiento. En este sentido, el estudio fue de tipo exploratorio.

Se hipotetiza que el uso de una herramienta de enseñanza de la programación, tal como los tutores cognitivos, influyen positivamente en el aprendizaje de los conceptos básicos de programación.

[30] Interesados en el tema pueden consultar (Kelleher, 2003)

Los tutores aquí descritos se encuentran en etapa de evaluación en forma de prototipos y los resultados obtenidos servirán para su posterior refinamiento.

MATERIALES Y MÉTODOS.

Tutores Cognitivos.

La literatura sobre herramientas de enseñanza basadas en software reporta un tipo especial de aplicación conocida como Tutor Cognitivo (J. R. Anderson et al., 1995), que cuenta con un historial positivo en la enseñanza de temas tales como el álgebra y la geometría (K. Koedinger & J. Anderson, 1997) y la enseñanza de lenguajes de programación de inteligencia artificial, tales como LISP (Corbett, 1993). La teoría detrás de estas herramientas es una teoría cognitiva conocida como ACT-R (Adaptive Control of Tought - Rational, J. Anderson, 2004; J.R. Anderson, 1996), que busca comprender y simular el funcionamiento de la mente humana.

Sin embargo, es reconocido que el costo de desarrollo de este tipo de aplicaciones es alto (K. Koedinger, V. Aleven, Hefferman, B. McLaren, & Hockenberry, 2004; Murray, 1999). Por ejemplo (K. Koedinger et al., 2004, pág. 8), mencionan que el esfuerzo estimado requerido para el desarrollo de un tutor inteligente se encuentra entre 100 y 1000 horas, por cada hora de instrucción.

En vista de tal problemática, algunos investigadores (V. Aleven, B. McLaren, J. Sewall, & K. Koedinger, 2006; Vincent Aleven, Bruce McLaren, Jonathan Sewall, & Keneth Koedinger, 2006), han buscado maneras de disminuir el tiempo y costo de desarrollo de los tutores cognitivos, eliminando el requerimiento de contar con conocimientos de modelación cognitiva y hasta cierto punto, de conocimientos de programación.

En el presente estudio, se utilizó la metodología propuesta por (K. Koedinger et al., 2004) en donde proponen las siguientes etapas para el desarrollo de tutores cognitivos, utilizando la herramienta CTAT[31] (Cognitive Tutors Authoring Tools) :

1. Crear una interfaz gráfica, a ser utilizada por el estudiante.

   Para esta investigación se utilizó el IDE Netbeans 5.0 para el desarrollo de la interfaz gráfica para el estudiante. Cabe mencionar que la herramienta CTAT incluye una serie de 'Widgets' (es decir, objetos) que permiten lograr

[31] http://ctat.pact.cs.cmu.edu/

interactividad con el aprendiz y monitorear el comportamiento de sus respuestas (ver Figura 1)

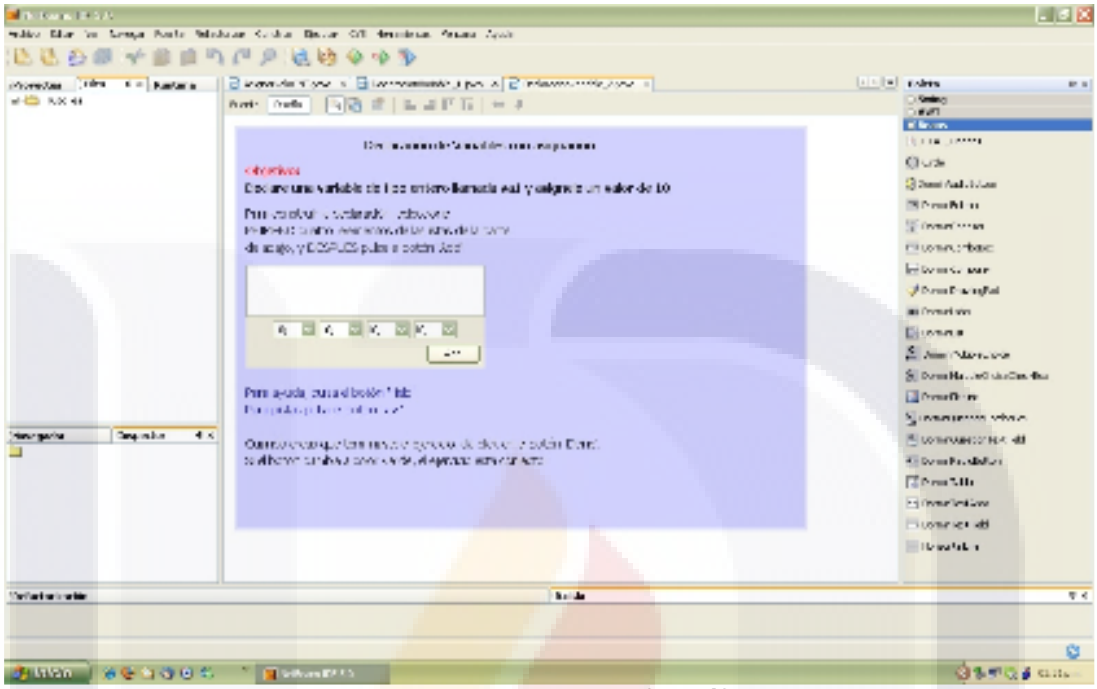


Figura 1. Desarrollo de interfaz gráfica.

2. Demostrar soluciones alternativas correctas e incorrectas.

Este paso consiste en que una vez diseñada la interfaz de estudiante, se procede a crear 'por demostración' las posibles alternativas o secuencias de acciones que el estudiante puede potencialmente seleccionar en la interfase.

3. Anotar los pasos de solución en un Árbol de Comportamiento (Behavior Graph).

La demostración del paso anterior, se registra en un árbol de comportamiento, anotando mensajes de ayuda, mensajes de retroalimentación y etiquetas para los conceptos y habilidades asociadas. Esta actividad es particularmente importante, ya que proporciona elementos de interactividad y retroalimentación al alumno (ver Figura 2). Los caminos correctos e incorrectos se señalan según corresponda y se anotan etiquetas a cada estado que corresponden a una habilidad concreta.

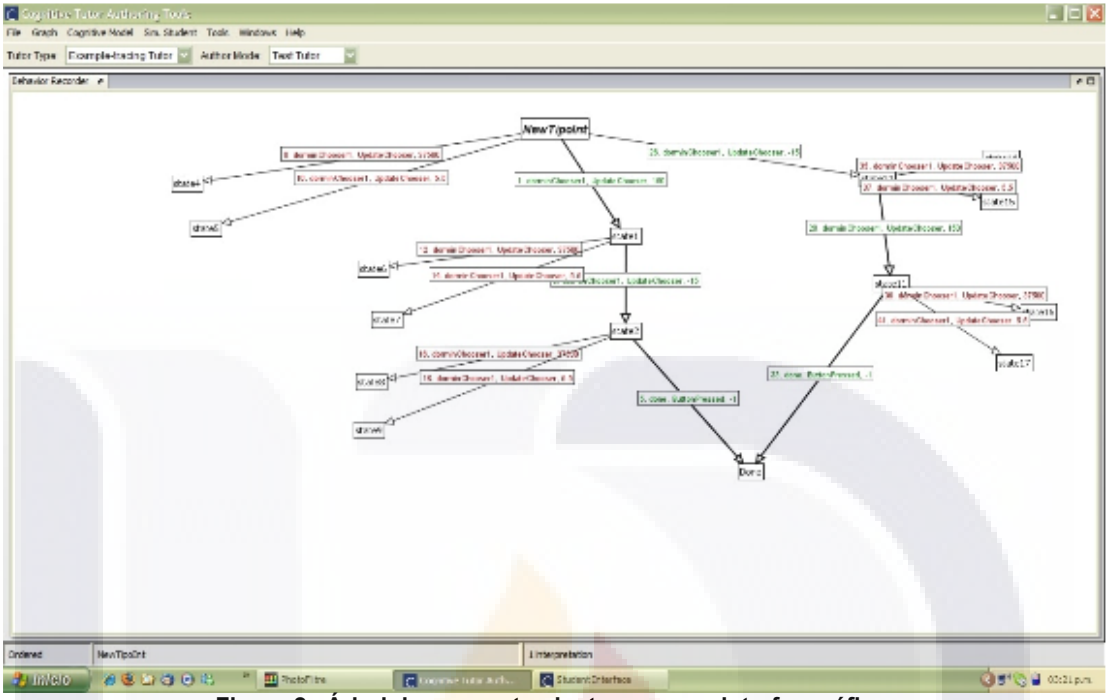


Figura 2. Árbol de comportamiento para una interfaz gráfica.

4. Inspeccionar y revisar la matriz de habilidades.

Los tutores desarrollados con la herramienta CTAT pueden ser de dos tipos: Tutores por demostración (Example-Trace Tutors) o Tutores Cognitivos. Los primeros sirven de prototipos para los segundos y se desarrollan por demostración sin necesidad de contar con muchos conocimientos de programación y modelación cognitiva. Para muchos casos, el tutor por demostración puede ser suficiente y conviene empezar por ahí, antes de desarrollar los tutores cognitivos, que por su parte requieren mayor esfuerzo de diseño. Una parte central de este diseño es la matriz de habilidades, que a su vez sirve de base para el desarrollo de las reglas de producción que darán un comportamiento inteligente y más general al tutor. Para este estudio, este paso no fue llevado a cabo, ya que los mini-tutores fueron desarrollados por demostración.

En el ámbito de la programación, los mini-tutores resultantes se enfocaron a la instrucción de los conceptos de tipo de dato entero, declaración y asignación de valores enteros en una sola sentencia (ver Figura 3). Se puso especial cuidado en los mensajes de retroalimentación y de ayuda al alumno (ver Figura 4)

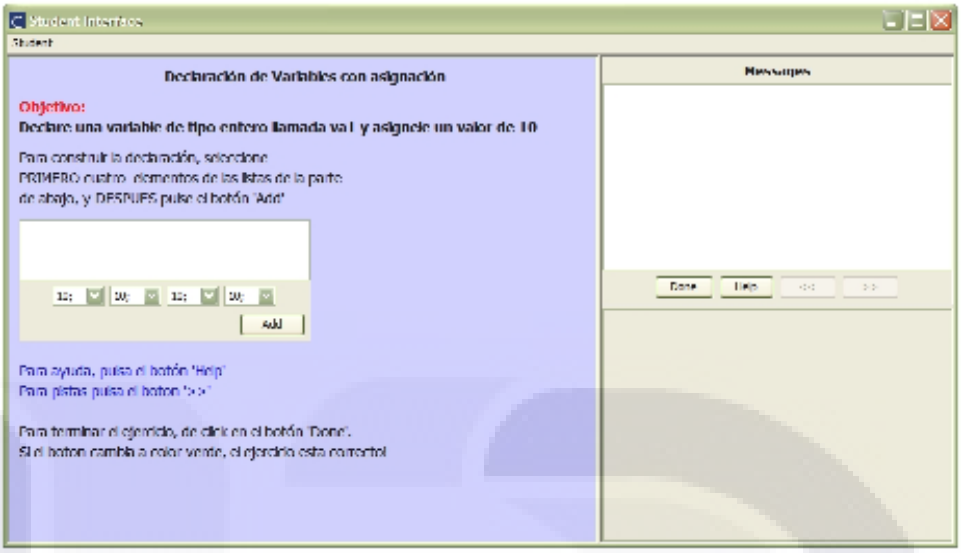


Figura 3. Ejemplo de Mini-tutor para el concepto de 'declaración y asignación de valores a variables enteras'

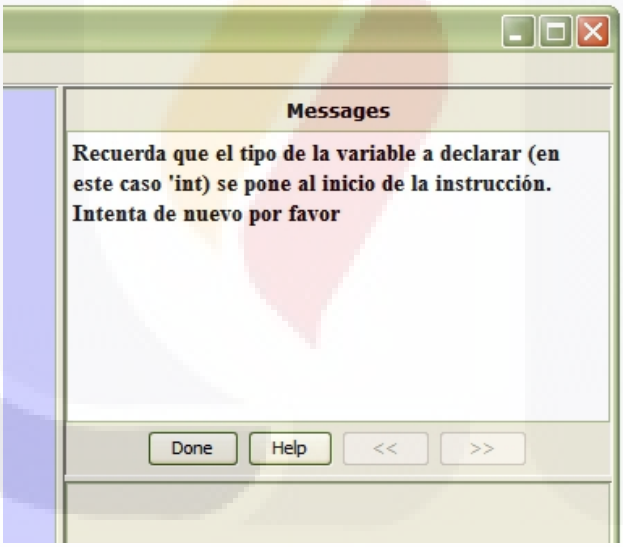


Figura 4. Ejemplo de mensaje de retroalimentación.

Para el despliegue de los mini-tutores se utilizó una plataforma LMS[32] (Learning Management System), en donde se diseñó un curso piloto (ver figura 5), simulando el estudio de un tema, en este caso, los tipos de datos y variables enteras en lenguaje C. Se desarrollaron lecciones textuales a manera de explicación teórica, presentadas a los participantes en forma de páginas Web

[32] Ésta se puede acceder en la dirección http://dsi.ccbas.uaa.mx/moodle

estáticas. Los mini-tutores cognitivos, se pusieron a disposición de los alumnos participantes al final de cada lección.

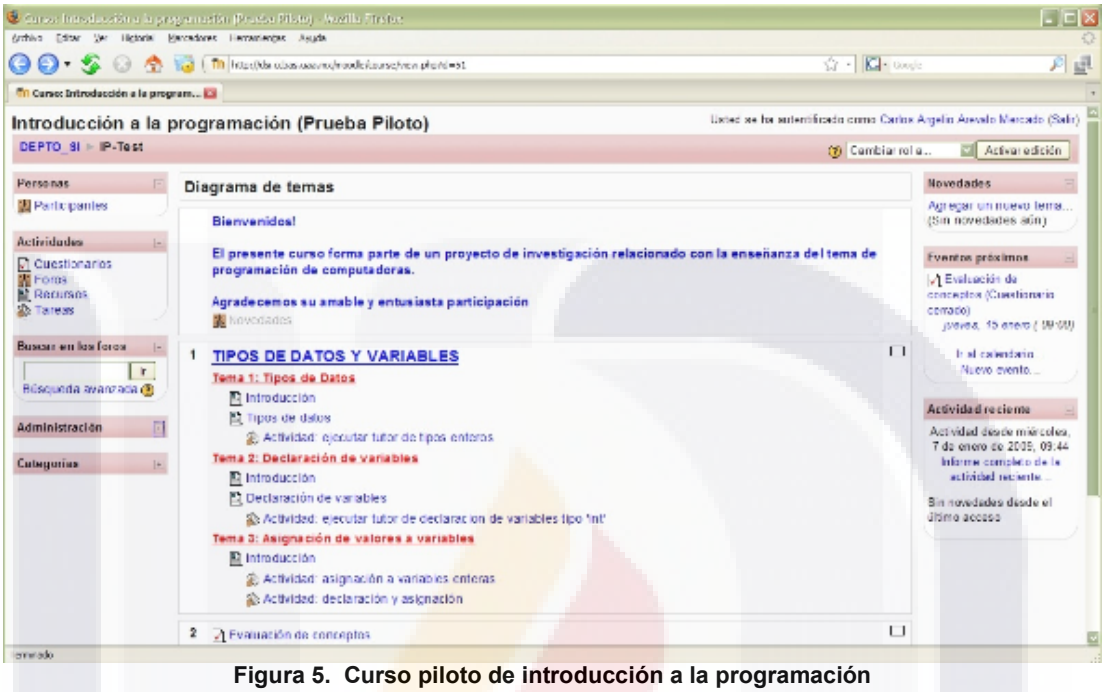


Figura 5. Curso piloto de introducción a la programación

Diseño Instruccional.

El diseño instruccional se define como "la ciencia de crear especificaciones detalladas para el desarrollo, evaluación y mantenimiento de situaciones que facilitan el aprendizaje de unidades temáticas tanto grandes como pequeñas" (Richey, 1986, pág. 9).

Rothwell y Kazanas, 1992 proponen un proceso de diseño instruccional (ver Figura 6), en el cual se busca enfocar el diseño del material didáctico (sea éste tradicional o basado en TI) hacia las necesidades específicas del aprendiz.

Sin embargo, Spiro, et. al. 2003, argumentan que hay una base común para el fracaso de muchos sistemas instruccionales, la cual tiene que ver con problemas básicos en el diseño del propio material de apoyo. R.J. Spiro, Feltovitch, Jacobson, y Coulson, (1991), señalan que los métodos de instrucción tradicionales que toman un enfoque lineal, no suelen tener problemas, cuando el material está bien estructurado y es de naturaleza simple. Pero en cambio "cuando el contenido aumenta en complejidad y poca estructuración, cantidades crecientes de información se pierden al usar los enfoques lineales y métodos de organización unidimensionales que tradicionalmente los acompañan" (R. Spiro et al., 1991, pág.

21). Por lo tanto, para que la instrucción sea efectiva, diversos tópicos altamente entrelazados deben considerarse de manera simultánea y no lineal.

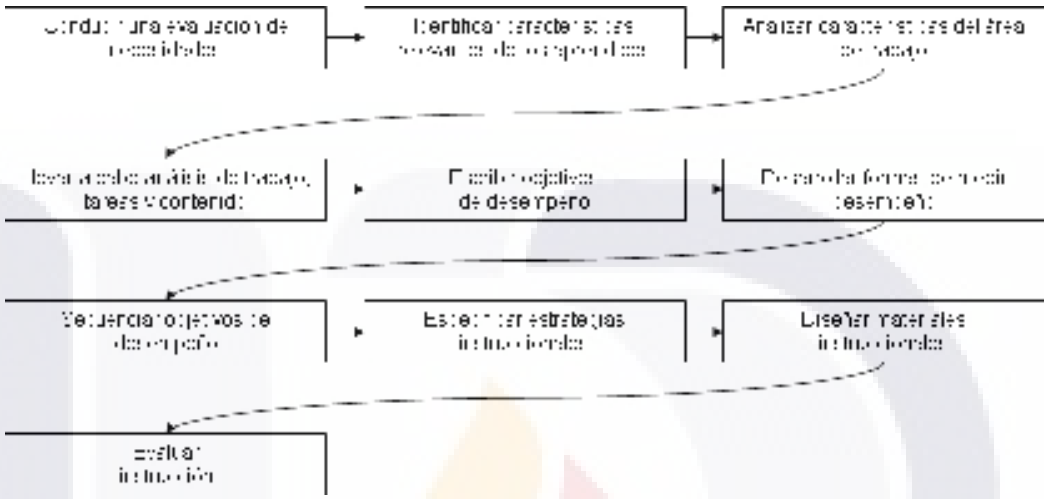


Figura 6. Proceso de Diseño Instruccional (Adaptado de Rothwell y Kazanas, 1992)

Para el presente estudio, las primeras tres etapas del proceso de diseño instruccional se llevaron a cabo mediante un estudio cualitativo, en la modalidad de sesión de Grupo Enfoque (Focus Group) a alumnos de tercer semestre[33] de la licenciatura en informática de la Universidad Autónoma de Aguascalientes, en el que por medio de análisis hermenéutico se detectó lo siguiente:

Tabla 1: Resultados de análisis hermenéutico, sesión de grupo de enfoque (Focus Group)

Tema relacionado	Ocurrencias
Interfase de usuario / usabilidad • Mensajes de error – retroalimentación • Idioma inglés • Exceso de características • Características didácticas de la herramienta • Visualización	9
Conocimiento cumulativo / conocimiento previo de la programación	7
Modelos mentales	5

[33] Notar que esta sesión de enfoque fue previa al estudio experimental realizado con alumnos de preparatoria. Estos datos cualitativos se recolectaron con el fin de entender cuales situaciones habían sido más problemáticas para los alumnos de primeros semestres de licenciatura en la carrera de informática de la UAA.

Estilo de enseñanza del profesor – material didáctico proporcionado	4
Estilo de aprendizaje del alumno	2
Dificultad en el cambio de sintaxis de un lenguaje a otro	3
Aprendizaje por descubrimiento	3

Las restantes actividades del proceso de diseño instruccional tienen que ver con un análisis del temático del dominio de conocimiento, en este caso, el aprendizaje de la programación. Un ejemplo simplificado de la descomposición temática del mismo puede visualizarse en el siguiente mapa conceptual (ver Figura 7).

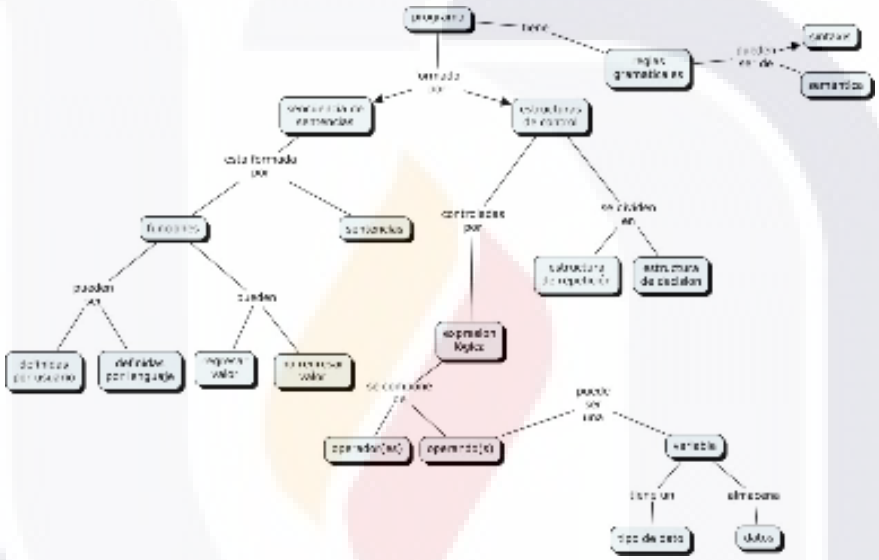


Figura 7. Ejemplo de una descomposición temática jerárquica de los conceptos básicos de programación

Si se toma en cuenta la interrelación que existe entre los conceptos del dominio (siguiendo las ideas indicadas por Rand J. Spiro & Collins, 2007), enfocándonos, por ejemplo, en el concepto de 'variable', podemos observar la siguiente situación de no-linealidad (ver Figura 8), en donde resalta la naturaleza compleja de los conceptos de programación, donde el concepto 'variable' se relaciona con otros conceptos, tales como funciones, expresiones lógicas, estructuras de control y parámetros, entre otros posibles.

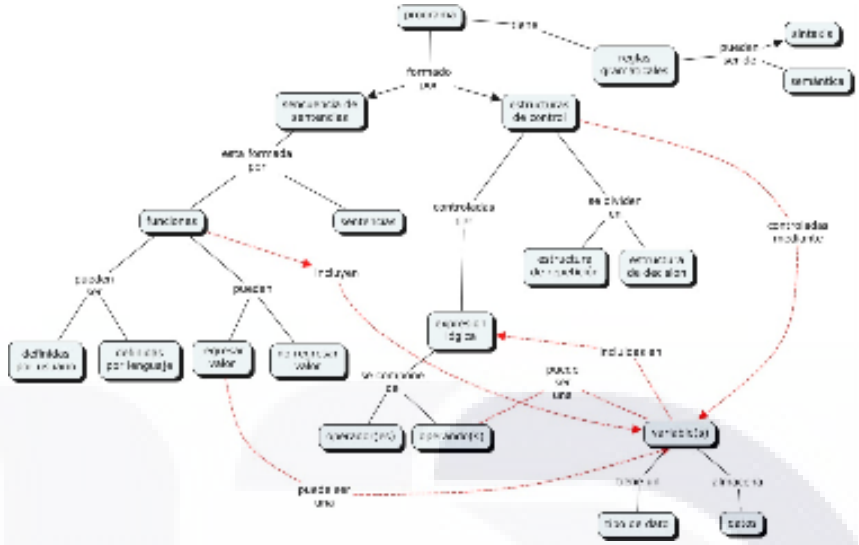


Figura 8. Ejemplo de una descomposición temática y jerárquica de los conceptos básicos de programación, incluyendo temas relacionados con el concepto 'variable'

Condiciones del estudio.

Se llevó a cabo un estudio cuasi-experimental con alumnos de preparatorias públicas sin experiencia previa en materias de programación (población objetivo), en donde se buscó medir el efecto en el desempeño de los mini-tutores en dos temas básicos de programación (los tipos de datos y la asignación de valores a variables). La selección de las preparatorias no se realizó de manera aleatoria, sino por disponibilidad de los participantes. La población de ambas preparatorias es similar en tanto provienen del sector público y por no contar con experiencia previa en la programación. La selección de los participantes fue por invitación.

Se crearon dos grupos con las siguientes características:

Grupo 1.
Alumnos de 5to semestre de la preparatoria pública Lic. Jesús Reyes Heroles (Aguascalientes), sin antecedentes de materias de programación. La cantidad de participantes en este grupo fue de 27 alumnos (n=27). Las condiciones ambientales consistieron en que el estudio se llevó a cabo en un laboratorio de cómputo con conexión a Internet. La participación del instructor consistió en indicar a los participantes donde se encontraban las ligas al material didáctico, en este caso, páginas Web estáticas. Este grupo no utilizó los mini-tutores, sirviendo como grupo de control.

Grupo 2.
Alumnos entre 1er y 5to semestre, de la Preparatoria de la Universidad Autónoma de Aguascalientes, sin experiencia previa de programación. Los alumnos participaron por invitación directa. Utilizaron computadoras y conexión a Internet

disponibles en sus domicilios. Para este grupo, la participación del instructor no fue presencial (a diferencia del grupo de control), pero se brindó asistencia en línea (vía Messenger) a los participantes que lo requirieron. Las instrucciones generales de acceso al curso piloto se les proporcionaron mediante un texto vía correo electrónico. Es importante mencionar que se registró el uso de los mini-turores por parte de los participantes de este grupo por medio del historial de actividades de la plataforma Moodle, en la cual se registran fechas y horas de entrada a cada recurso, corroborando así su uso. No se incluyeron instrucciones de uso sobre los mini-tutores. El tamaño de la muestra fue de 19 (n=19).

Instrumento de evaluación.

Se utilizó un cuestionario de ocho preguntas de opción múltiple, con un tiempo límite para ambos grupos de 15 minutos, que evaluaba los conceptos de:

- Tipo de dato 'int' (entero)
- Declaración de variables int
- Tipos de datos char (carácter)
- Declaración de variables char

El diseño del instrumento se realizó tomando como referencia los conceptos explicados en el propio material didáctico en línea. El instrumento se aplicó igualmente en línea a ambos grupos (en forma de cuestionario de opción múltiple), por medio de la plataforma Moodle. Esto permitió registrar la duración en el llenado del cuestionario de cada participante y realizar análisis de correlación. En el caso del Grupo 1, el llenado fue al final de la sesión de laboratorio y en el caso del Grupo 2, al finalizar todas las sesiones y después de haber usado los mini-tutores. El instrumento no fue previamente validado para verificar el comportamiento normal del mismo, lo que puede considerarse como una limitación del estudio.

RESULTADOS

Se recolectaron 46 observaciones entre los dos grupos participantes. Se descartaron cuatro observaciones por contener datos inconsistentes[34]. Los resultados generales, indicados por la estadística descriptiva de las observaciones recolectadas para ambos grupos se muestran en la Tabla 2:

Tabla 2. Estadística descriptiva

		Tiempo_Gpo1	Calificación_Gpo1	Tiempo_Gpo2	Calficación_Gpo2
N	Válidos	27	27	19	19

[34] Cuestionarios que no fueron terminados y que registraron una calificación de '0'

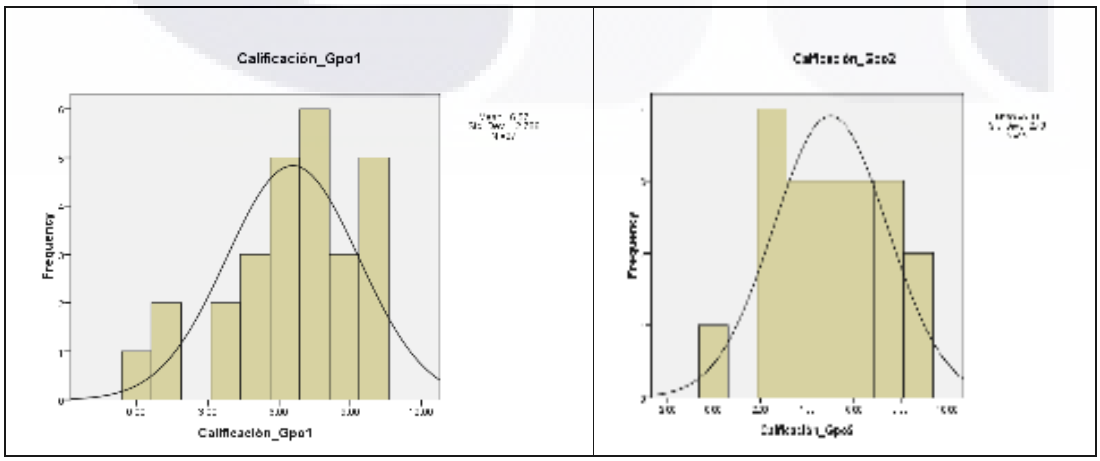
Faltantes	0	0	8	8
Media	3.6841	6.5741	3.6500	5.0000
Mediana	3.0000	7.5000	3.0000	5.0000
Moda	3.00	7.50	1.00[a]	2.50
Desviación Std.	2.15519	2.78631	3.51185	2.42956
Varianza	4.645	7.764	12.333	5.903
Mínimo	.47	.00	.35	.00
Máximo	9.00	10.00	15.00	8.75

a. Existen múltiples modas. Se muestra el valor más pequeño.

Se observa que la media de la calificación del Grupo 1 es mayor (6.57) a la del Grupo 2 (5.0). Es decir, que en promedio la calificación del grupo experimental fue menor a la del grupo de control. En ambos casos, la desviación estándar es muy similar, lo que indica poca variabilidad y dispersión de los datos.

La distribución de frecuencias de las calificaciones obtenidas por ambos grupos (ver Tabla 3), muestra que el Grupo 1 (grupo de control sin uso de tutores cognitivos) presenta el mayor porcentaje de frecuencias en el rango de 7.5 de calificación. Muestra además 5 observaciones con calificación de 10. El Grupo 2 (grupo experimental usando tutores cognitivos a distancia) presenta una distribución de porcentajes uniforme (15.8%) en los rangos de calificaciones de 3.75, 5, 6.25 y 7.5. Este grupo no tuvo calificaciones iguales a 10.

Tabla 3. Distribuciones de frecuencias de las calificaciones obtenidas por ambos grupos



Calificación Grupo 1

		Frec.	Porc.	Porcentaje Válido	Porcentaje Acumulado
Válidos	0	1	3.7%	3.7%	3.7%
	1.25	2	7.4%	7.4%	11.1%
	3.75	2	7.4%	7.4%	18.5%
	5	3	11.1%	11.1%	29.6%
	6.25	5	18.5%	18.5%	48.1%
	7.5	6	22.2%	22.2%	70.4%
	8.75	3	11.1%	11.1%	81.5%
	10	5	18.5%	18.5%	100.0%
	Total	27	100.0%	100.0%	

Calificación Grupo 2

		Frec.	Porc.	Porcentaje Válido	Porcentaje Acumulado
Válidos	0	1	5.3%	5.3%	5.3%
	2.5	4	21.1%	21.1%	26.3%
	3.75	3	15.8%	15.8%	42.1%
	5	3	15.8%	15.8%	57.9%
	6.25	3	15.8%	15.8%	73.7%
	7.5	3	15.8%	15.8%	89.5%
	8.75	2	10.5%	10.5%	100.0%
	Total	19	100.0%	100.0%	

Dados los resultados anteriores, podría interpretarse que el Grupo 1 tuvo un mejor desempeño que el Grupo 2. Para verificar si existió una diferencia estadísticamente significativa en las calificaciones de ambos grupos, se corrió una prueba de análisis de varianza (ver Tabla 4), resultando un valor de 0.053, que proporciona indicios de que no existe diferencia en el desempeño de ambos grupos.

Tabla 4. Prueba ANOVA

Calificación

	Suma de cuadrados	df	Cuadrado de la media	F	Sig.
Entre grupos	27.632	1	27.632	3.946	0.053
Dentro de los grupos	308.102	44	7.002		
Total	335.734	45			

Las pruebas de correlación no arrojan un valor significativo (.031) entre tiempo y calificación (ver Tabla 5). Es decir, no hay una correlación entre el desempeño de los participantes y el tiempo tomado en contestar la evaluación.

Tabla 5. Pruebas de correlación

		Tiempo	Calificación
Tiempo	Correlación Pearson	1.000	.031
	Sig. (2-tailed)		.838
	N	46	46
Calificación	Correlación Pearson	.031	1.000
	Sig. (2-tailed)	.838	

Tabla 5. Pruebas de correlación

		Tiempo	Calificación
Tiempo	Correlación Pearson	1.000	.031
	Sig. (2-tailed)		.838
	N	46	46
Calificación	Correlación Pearson	.031	1.000
	Sig. (2-tailed)	.838	
	N	46	46

DISCUSIÓN

A primera vista, los resultados obtenidos en cuanto al efecto en el aprendizaje pueden parecer desalentadores, pero creemos que estos son preliminares y es necesario realizar más estudios experimentales, considerando los resultados de otros estudios (Aleven et al., 2006; J. R. Anderson et al., 1995; Kumar, 2006), en donde se reportan efectos positivos. Puede además agregarse que la experiencia obtenida sobre las variables experimentales, resultó útil y será tomada en cuenta para futuros desarrollos.

CONCLUSIONES

Dadas los resultados de las pruebas estadísticas realizadas, se considera que los resultados obtenidos en el estudio no permiten concluir aún si los tutores cognitivos para la enseñanza de la programación, desarrollados con las características permitidas por la herramienta CTAT utilizada, tienen un efecto positivo en el aprendizaje de estudiantes de nivel medio, debido a variaciones no previstas en las condiciones de las muestras.

Por otro lado, se considera que los objetivos secundarios del estudio si fueron alcanzados, al detectarse la naturaleza de dichas variaciones e identificar como éstas pueden influir en el aprendizaje.

Por ejemplo, se observó que el instructor –que no estuvo presente en el grupo experimental–, pudo haber influido en la confianza y facilidad de uso del material didáctico de los participantes del grupo de control. Es decir, que a la hora de asistir en las dudas sobre el uso del material (aún cuando este consistió solo de páginas Web estáticas) los participantes del grupo de control pudieron recurrir a un instructor humano que estaba disponible en el laboratorio donde se condujo el

estudio. Esto mostró indicios de que el diseño del tutor cognitivo debe incluir suficientes características de 'ayuda' para el alumno.

También, puede argumentarse que los participantes del grupo experimental pudieron haberse beneficiado al haber recibido una capacitación previa, ya sea presencial o a distancia. Es decir, aún cuando el diseño de la interfaz de usuario de los mini-tutores cognitivos es muy sencilla, posiblemente las instrucciones enviadas por escrito no fueron suficientes. En este sentido, los resultados sugieren que la aplicación de pruebas de usabilidad pueden brindar retroalimentación a los desarrolladores (e instructores) sobre el diseño de la interfaz de los mini-tutores cognitivos.

Finalmente, algunas de las limitantes técnicas (por ejemplo, la necesidad de instalar componentes de software en las máquinas de algunos participantes) o demográficas encontradas (por ejemplo, la diferencia de semestres en una parte de los participantes del Grupo 2), creemos han aportado una experiencia valiosa para la ejecución de subsecuentes experimentos.

REFERENCIAS

ALEVEN, V., MCLAREN, B., SEWALL, J., & KOEDINGER, K., Rapid authoring of intelligent tutors for real-world and experimental use, En *9th International Conference on Intelligent Tutoring Systems (ITS 2006)*. Presented at the International Conference on Intelligent Tutoring Systems (ITS 2006), Berlin, págs. 61-70, 2006.

ALEVEN, V., MCLAREN, B., SEWALL, J., & KOEDINGER, K., The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. En *7th Annual Intelligent Tutoring Systems Conference*. Presented at the Intelligent Tutoring Systems Conference, Maceio, Brazil, 2005.

ANDERSON, J., An Integrated Theory of the mind. *Psychological Review*, 111(4), 1036-1060, 2004.

ANDERSON, J. R., CORBET, A., & KOEDINGER, K. R. Cognitive Tutors: Lessons Learned, *The Journal of Learning Sciences*, 4(2), 167-207, 1995.

ANDERSON, J., ACT: A simple Theory of Cognition, *American Psychologist*, 51(4), 355-365, 1956.

BANDURA, A., Self-Efficacy Mechanism in Human Agency, *American Psychologist*, 37(2), 122-147, 1982.

BAYMAN, P., A diagnosis of beginning programmers' misconceptions of BASIC programming statements, *Communications of the ACM*, 26(9), 677 - 679, 1983.

BLOOM, B. S., The 2 sigma problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring, *Educational Researcher*, 13(6), 4-16, 1984.

DU BOULAY, B., Can we learn from ITSs?, En *Intelligent Tutoring Systems*, Lecture notes in computer science (Vol. 1839), Springer Berlin / Heidelberg, págs. 9-17, 2000.

BOYLE, T., The design and development of second generation learning objects, En *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2005*, Chesapeake, VA., págs. 2-12, 2006.

BROSNAN, M., The impact of computer anxiety and self-efficacy upon performance, *Journal of Computer Assisted Learning*, 14, 223-234, 1998.

BRUSILOVSKY, P., Worlds-languages: A Way to Learn Programming Principles, *Education and Information Technologies*, 2(1), 65-83, 1996.

BRUSILOVSKY, P., Intelligent learning environments for programming: The case for integration and adaptation, , En *Proceedings of AI-ED'95. 7th World Conference on Artificial Intelligence in Education*, Washington, DC., (págs. 1-8), 1995.

CHANSILP, K., Student's responses to the use of an interactive multimedia tool for learning computer programming, En *World Conference on Educational Multimedia, Hypermedia & Telecommunications*, Proceedings of Ed-Media 2004, Norfolk, USA: L. Cantoni & C. McLoughlin (Eds)., (págs. 1739-1746), 2004.

COOPER, S., Using animated 3D graphics to prepare novices for CS1, *Computer science education*, 13(1), 3-30, 2003.

CORBETT, A., The predictive validity of student modeling in the ACT Programming Tutor, *Artificial Intelligence and Education: The Proceedings of AI-ED 93*. Charlottesville, VA: AACE. In P. Brna, S. Ohlsson, & H. Pain (Eds.), 1993.

DIJKSTRA, E., On the Cruelty of Really Teaching Computer Science, *Communications of the ACM*, 32(12), 1398-1404, 1989.

DU BOULAY, J., Some difficulties of learning to program, *Lawrence Erlbaum Associates, Hillsdale*, 1989.

FIX, V., , Mental representations of programs by novices and experts, En *Conference on Human Factors in Computing Systems, Proceedings of the SIGCHI conference on Human factors in computing systems*, Amsterdam, The Netherlands : ACM Press New York, NY, USA., págs. 74-79, 1993.

HAGAN, D., & MARKHAM, S., ¿Does it help to have some programming experience before beginning of a computer degree program?, En *Annual Joint Conference Integrating Technology into Computer Science Education*, Proceedings of the 5th annual SIGCSE/SIGUE ITICSEconference on Innovation and technology in computer science education, Helsinki, Finland (págs. 25 - 28), 2000.

HAMER, J., A lightweight visualizer for Java, En *Proceedings of the Third Program Visualization Workshop*, Warwick, UK, (págs. 64-68), 2004.

HEGARTY, M., Constructing mental models from text and diagrams, *J. Mem. Lang.*, 32, 717–742, 1993.

HEGGESTAD, E. The Predictive Validity of Self-Efficacy in Training Performance: Little More Than Past Performance. *Journal of Experimental Psychology*: 11(2), 94-97, 2006.

HOLDEN, E., y WEEDEN, E. Prior Experience and New IT Students, Issues in *Informing Science and Information Technology*, 2, 169, 2005.

HU, C., It's Mathematical: after all -the nature of learning computer programming. *Educational Information Technology*, 11, 83-92. doi: 10.1007/s10639-005-5714-4, 2006.

JEHNG, J., A visualisation approach to learning the concept of recursion. *Journal of Computer Assisted Learning*, 15, 279-290, 1999.

JENKINS, T., On the difficulty of learning to program, En *3rd annual LTSN-ICS Conference*. Loughborough University. LTSN Centre of information and computer sciences, 2002.

KELLEHER, C., *Lowering the Barriers to Programming: a survey of programming environments and languages for novice programmers* (págs. 83-137), 2003.

KOEDINGER, K., ALEVEN, V., HEFFERNAN, N., MCLAREN, B., & HOCKENBERRY, M. Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration, En *Proceedings of 7th Annual Intelligent Tutoring Systems Conference*. Maceio, Brazil, 2004.

KOEDINGER, K., y ANDERSON, J. Intelligent Tutoring Goes To School in the Big City. *International Journal of Artificial Intelligence in Education*, 8, 30-43, 1997.

KUJANSUU, E., Using program visualisation learning objects with non-major students with different study background, En *Methods, Materials and Tools for Programming Education*, Tampere, Finland, (págs. 21-26), 2006.

KUMAR, A. N., The Effect of Using Problem-Solving Tutors on the Self-Confidence of Students. *Presented at the 18th Workshop of the Psychology of Programming Interest Group*, University of Sussex, (págs. 275 - 283), 2006.

LAI, L., Investigating the viability of mental models held by novice programmers. *ACM SIGCSE Bulletin*, 39(1), 499-503, 2007.

MATTHÍASDÓTTIR, Á., Usefulness of learning objects in computer science learning. En *Methods, Materials and Tools for Programming Education*. Tampere, Finland: TAMPERE POLYTECHNIC - University of Applied Sciences Publications, (págs. 27-31), 2006.

MCIVER, L., GRAIL: A Zeroth Programming Language, En *International conference of computing in education (ICCE)*. Oita, Japan, (págs. 43-50), 1999.

MCKEOWN, J., The use of a multimedia lesson to increase novice programmers' understanding of programming array concepts. *Journal of Computing Sciences in Colleges*, Volume 19(4), 39 - 50, 2004.

MILNE, I., & ROWE, G., Difficulties in Learning and Teaching Programming—Views of Students and Tutors. *Education and Information Technologies*, 7(1), 55-66, 2002.

MOISEY, S., Fulfilling the promise of learning objects. En *In Handbook of Distance Education* (Vol. 1, pág. 323). Lawrence Erlbaum, 2003.

MURRAY, T., Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art, *International Journal of Artificial Intelligence in Education*, 10, 98-129, 1999.

NARS, T., A Java visualizer class: incorporating algorithm visualizations into students' programs. En *Annual Joint Conference Integrating Technology into Computer Science Education , Proceedings of the 5th annual conference on the teaching of computing and the 3rd annual conference on integrating technology into computer science education: Changing the delivery of computer science education*, Dublin City Univ., Ireland : ACM Press New York, NY, USA, (págs. 181 - 184), 1999.

NEVEN, F., Reusable learning objects: a survey of LOM-based repositories. En *International Multimedia Conference, Proceedings of the tenth ACM international conference on Multimedia*, Juan-les-Pins, France, (págs. 291 - 294), 2002.

RAMALINGAM, V., Self-Efficacy and mental models in learning to program. En *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, Leeds, United Kingdom : ACM Press New York, NY, USA, (págs. 171 - 175), 2004.

RICHEY, R., *The Theoretical and Conceptual Bases of Instructional Design*. New York: Nichols, 1986.

ROTHWELL, W. J., & KAZANAS, H., *Mastering the instructional design process: a systematic approach* (First Edition.). San Francisco: Jossey-Bass Publishers.

SADLER-SMITH, E., & SMITH, P. J., Strategies for accommodating individuals' styles and preferences in flexible learning programmes, *British Journal of Educational Technology*, *35*(4), 395–412, 2004.

SPIRO, R., COLLINS, B. P., THOTA, J. J., & FELTOVITCH, P. J., Cognitive Flexibility Theory: Hypermedia for Complex Learning, Adaptive Knowledge Application, and Experience Acceleration, *Educational Technology*, *43*(5), 5-10, 2003.

SPIRO, R., FELTOVITCH, P., JACOBSON, M., & COULSON, R., Cognitive Flexibility, Constructivism, and Hypertext: Random Access Instruction for Advanced Knowledge Acquisition in Ill-Structured Domains, *Educational Technology*, 24-33, 1991.

SPIRO, R. J., & COLLINS, B. P., Reflections on a Post-Gutenberg Epistemology for Video Use in Ill-Structured Domains: Fostering Complex Learning and Cognitive Flexibility, En *Video research in the learning sciences*. Lawrence Erlbaum Associates, 2007.

THOMAS, L., RATCLIFFE, M., WOODBURY, J., & JARMAN, E., Learning Styles and Performance in the introductory programming sequence, *ACM SIGCSE Bulletin*, *34*(1), 33 - 37, 2002.

WHITE, G., Standarized mathematics scores as a prerequisite for a first programming course, *Mathematics and Computer Education*, *37*(1), 2003.

WIEDENBECK, S., LABELLE, D., & KAIN, V. Factors affecting course outcomes in introductory programming, En *16th Workshop of the psychology of programming interest group*. Institute of Technology, Carlow, Ireland, 2004.

WILEY, D., *The Instructional Use of Learning Objects*. Bloomington, IN: Association for Educational Communications and Technology, 2000.

7 GLOSARIO.

Artefacto	Es una innovación que define las ideas, prácticas, habilidades técnicas y productos a través de los cuales el análisis, diseño, implementación y uso de sistemas de información pueden ser efectiva y eficientemente alcanzados. Pueden ser constructos, modelos, métodos e instanciaciones.
Ejemplo resuelto	Enfocan la atención del estudiante hacia los diversos estados de un problema y sus pasos de solución, habilitando en los aprendices, por un proceso de inducción, la adquisición de estrategias de solución de problemas no generales.
Interfaz de usuario	Es la parte del software visible para el usuario y es uno de los aspectos más importantes a diseñar correctamente. Un mal diseño en este componente puede provocar que los usuarios descarten el uso del software.
ISDT	Ver Teoría de diseño de sistemas de información.
Maquina nocional	El modelo mental que tiene el usuario sobre la computadora cuando ésta ejecuta programas, y que le sirve para contestar la pregunta ¿qué tipo de comandos son los que entiende la computadora? Esta máquina nocional se crea respecto al lenguaje de programación.
Metacognición	La conciencia sobre como la propia persona aprende. A través de la metacognición el individuo puede definir la naturaleza de un problema o tarea, seleccionar una representación física o mental útil, seleccionar la estrategia más efectiva para ejecutarla, activar conocimiento previo relevante, prestar atención a la retroalimentación sobre cómo se está avanzando en la resolución del problema o tarea.

Meta-diseño	El meta-diseño describe las características de una clase de artefacto de software que se hipotetiza va a satisfacer los meta-requerimientos.
Modelo Mental	Es la representación interna de una tarea o sistema complejo, cuya construcción permite al aprendiz el razonar, predecir y comprender el funcionamiento de tal tarea o sistema.
Problema por completar	Los problemas por completar son problemas en los que se cuenta con un estado de avance parcial y un estado objetivo deseado y se solicita a los aprendices que provean una o varias soluciones parciales o intermedias.
Programación	Programar consiste en idear un proceso de solución a un problema, combinando un conjunto limitado de estructuras lógicas predefinidas, mediante un lenguaje de programación.
Protocolo verbal	Los protocolos verbales documentan el comportamiento mental de una persona al solicitársele que “hable en voz alta” mientras lleva a cabo una tarea en particular. Posteriormente estos documentos transcritos se analizan para estudiar el proceso de pensamiento seguido por la persona.
Prototipo evolutivo	El prototipo evolutivo inicia como un sistema sencillo que incluye los requerimientos más importantes del usuario y se va aumentando o cambiando según se descubren nuevos requisitos para finalmente convertirse en el sistema solicitado por tal usuario.
Teoría de codificación dual	Propone que el sistema de memoria de corto plazo humano consta de dos grandes subsistemas: uno verbal y uno visual. El subsistema visual procesa y almacena información concreta, tal como imágenes y sonidos. El subsistema verbal se encarga de procesar el lenguaje y la información abstracta. De acuerdo con esta teoría, ambos sistemas son independientes, pero conectados entre sí. A la creación ya

	sea de una representación verbal a partir de un estímulo visual, o de una imagen a partir de lenguaje, se le conoce como conexión referencial
Teoría de diseño de sistemas de información	Tiene dos componentes básicos: producto y proceso. El primer aspecto se enfoca hacia las características de una clase de artefacto de software (producto de diseño) para un tipo especial de problema y el segundo hacia un proceso de diseño sugerido para construir tal clase de artefacto (proceso de diseño).
Teoría Núcleo	Teorías tomadas de todas las ramas de la ciencia y se supone gobiernan los requerimientos de diseño.
Transferencia	En aspectos de aprendizaje, la transferencia tiene lugar cuando se usa lo aprendido para resolver nuevos problemas, contestar nuevas preguntas o facilitar el aprendizaje de nuevos temas. Está asociada al aprendizaje significativo.
Usabilidad	La usabilidad no es una propiedad unidimensional de las interfaces de usuario, sino que tiene múltiples componentes, asociados por lo general a cinco grandes atributos: facilidad de aprendizaje, eficiencia, facilidad de recordar, disminución de errores y satisfacción.

8 BIBLIOGRAFIA

Agre, P.E. 1997. *Computation and Human Experience*. Cambridge: Cambridge University Press.

Aleven, V., B. McLaren, J. Sewall, y K. Koedinger. 2006. Rapid authoring of intelligent tutors for real-world and experimental use. En *8th International Conference on Intelligent Tutoring Systems (ITS 2006)*, 61-70. Berlin.

Anderson, J., D. Bothell, y M. Byrne. 2004. An Integrated Theory of the mind. *Psychological Review* 111, nº. 4: 1036-1060.

Anderson, J., R. Farrell, y R. Sauers. 1984. Learning to program in LISP. *Cognitive Science*.

Anderson, J. R., C. Boyle, y A. Corbell. 1990. Cognitive modelling and intelligent tutoring. *Artificial Intelligence* 42: 7-49.

Anderson, J. R., A.T. Corbet, y K. R. Koedinger. 1995. Cognitive Tutors, Lessons Learned. *The Journal of Learning Sciences* 4, nº. 2: 167-207.

Anderson, J. R., J.M. Fincham, y S. Douglass. 1997. The role of examples and rules in the acquisition of a cognitive skill. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 23: 932-945.

Anderson, J.R. 1996. ACT, A simple Theory of Cognition. *American Psychologist* 51, nº. 4 (Abril): 355-365.

Andriole, S.J. 1992. *Rapid application prototyping (2nd ed.)*. Wellesley, MA, USA: QED Information Sciences, Inc. http://portal.acm.org/citation.cfm?id=127783.

Arévalo, C., y J. Gómez. 2010. Un estudio piloto sobre el efecto de los tutores cognitivos para la enseñanza de conceptos básicos de programación. *Investigación y Ciencia de la Universidad Autónoma de Aguascalientes* 48.

Arévalo, C., N. Hernández, y J. Gómez. 2007. Uso de métricas de software en el contexto del debate de la enseñanza de la programación orientada a objetos como primer paradigma. En *Proceedings 22nd Annual Conference of the International Academy of Information Management, Special Interest Group on Education*. Montreal, Quebec, Canadá.

Arevalo, Carlos, Lizbeth Andrade, y Juan Gómez. 2008. The effect of learning objects on a C++ Programming Lesson. En *19th Annual Conference of the International Information Management Association*. San Diego, CA. http://www.iima.org/Proceedings/Proceedings-13-15Oct2008.html.

Arshad, Naveed. 2009. Teaching Programming and Problem Solving to CS2 Students using Think-Alouds. *ACM SIGCSE Bulletin* 41, nº. 1: 372-376.

Ausubel, D.P. 1963. *The psychology of meaningful verbal learning*. New York: Grune and Stratton.

———. 1968. *Educational psychology: A cognitive view*. New York: Holt, Rinehart and Winston.

Avello, D.G. 2003. Reflexiones y experiencias sobre la enseñanza de POO como único paradigma. En *JENUI 2003 - IX Jornadas de Enseñanza Universitaria de la Informática.* Cádiz, Spain.

Bandura, A. 1982. Self-Efficacy Mechanism in Human Agency. *American Psychologist* 37, nº. 2: 122-147.

Bayman, P. 1983. A diagnosis of beginning programmers' misconceptions of BASIC programming statements. *Communications of the ACM* 26, nº. 9 (Septiembre): 677 - 679.

Ben-Ari, Mordechai. 1998. Constructivism in Computer Science Education. *ACM SIGCSE Bulletin*.

Benbasat, I., y R. Zmud. 2003. The Identity Crisis Within the IS Discipline: Defining and Communicating the Discipline's Core Properties. *MIS Quarterly* 27, nº. 2: 183-194.

Bereiter, C., y M. Scarmadalia. 1985. Cognitive coping strategies and the problem of "inert knowledge". En *Thinking and learning skills: research and open question*, 2:65-80. New Jersey: Erlbaum.

Bergin, S. 2005. Programming: factors that influence success. En *SIGCSE '05*, February:23-27. St. Louis Missouri, USA: ACM.

Bishop-Clark, C. 1995. Cognitive Style, Personality, and computer programming. *Computers in human behavior* 11: 241-260.

Boehm, B., y T. Gray. 1984. Prototyping versus specifying: A multi-project experiment. *IEEE Transactions on software engineering* SE-10, nº. 3: 290-303.

Booth, S. 2001. Learning to program as entering the datalogical culture: a phenomenographic exploration. En *9th EARLI conference*. Fribourg Switzerland, August 2001.

Boyle, T. 2003. Improved success rates for students studying Programming. *Investigations in university teaching and learning* 1, nº. 1: 52-54.

Brooks, R.E. 1977. Towards a theory of the cognitive processes in computer programming. *International Journal of Man-Machine Studies* 9: 737–751.

———. 1983. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies* 18: 543–554.

———. 1990. Categories of programming knowledge and their application. *International Journal of Man-Machine Studies* 18: 543-554.

Brosnan, M.J. 1998. The impact of computer anxiety and self-efficacy upon performance. *Journal of Computer Assisted Learning* 14 (Marzo): 223-234.

Bruce, C. 2004. Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education* 3: 144.

Brusilovsky, P. 1998. Mini-languages: A Way to Learn Programming Principles. *Education and Information Technologies* 2, nº. 1: 65-83.

Burkhardt, Jean-Marie, Françoise Détienne, y Susan Wiedenbeck. 1997. Mental Representations Constructed by Experts and Novices in Object-Oriented Program Comprehension. *Dans Human-Computer Interaction*.

Byrne, P. 2001. The effect of student attributes on success in programming. *ACM SIGCSE Bulletin* 33, nº. 3: 49 - 52.

Caspersen, Michael E., y Jens Bennedsen. 2007. Instructional design of a programming course: a learning theoretic approach. En *Proceedings of the third international workshop on Computing education research*, 111 - 122.

Atlanta, Georgia, USA: ACM New York, NY, USA.
Chansilp, K.O. 2004. Student's responses to the use of an interactive multimedia tool for learning computer programming. En *World Conference on Educational Multimedia, Hypermedia & Telecommunications*, 1739-1746. Proceedings of Ed-Media 2004. Norfolk, USA: L. Cantoni & C. McLoughlin (Eds).
Chen, P. 1976. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)* 1, nº. 1: 9-36.
Chi, M., M. Bassok, M. Lewis, P. Reimann, y R. Glaser. 1982. Expertise in problem solving. En *Advances in the Psychology of Human Intelligence*, 7-75. Hillsdale, NJ: Erlbaum.
Chieu, Vu M. 2007. COFALE: An Authoring System for Creating Web-based Adaptive Learning Environments Supporting Cognitive Flexibility. *Journal of Computers* 2, nº. 5 (Julio): 27-37.
Conway, Matthew J. 1997. Alice:
Easy-to-Learn 3D Scripting for Novices. PhD. Thesis, University of Virginia, Diciembre.
Cooper, G., y J. Sweller. 1987. The effects of schema acquisition and rule automation of mathematical problem-solving transfer. *Journal of Educational Psychology* 79: 347--362.
Corbett, A. 1993. The predictive validity of student modeling in the ACT Programming Tutor. En . Artificial Intelligence and Education: The Proceedings of AI-ED 93. Charlottesville, VA: AACe.: In P. Brna, S. Ohlsson, & H. Pain (Eds.).
Dahlbom, B. 1996. The New Informatics. *Scandinavian Journal of Information Systems* 8, nº. 2: 29-47.
Date, C.J. 1990. *An introduction to database systems*. 5º ed. Addison-Wesley Systems Programming Series.
http://portal.acm.org/citation.cfm?id=SERIES9108.77706.
Davies, S.P. 1993. Models and Theories of programming strategy. *International Journal of Man-Machine Studies*.
Davis, A. 1992. Operational Prototyping: A New Development Approach. *IEEE Software* 9, nº. 5: 70-78.
Dehnadi, Saeed, y Richard Bornat. 2006. *The camel has two humps*. Middlesex University. http://www.cs.mdx.ac.uk/research/PhDArea/saeed.
Détienne, Françoise. 1995. Design Strategies and Knowledge in Object-Oriented Programming: Effects of Experience. *Human-Computer Interaction* 10: 129-170.
Dijkstra, E. 1989. On the Cruelty of Really Teaching Computer Science. *Communications of the ACM* 32, nº. 12 (Diciembre): 1398-1404.
Du Boulay, J.B.H. 1989. *Some difficulties of learning to program.* Lawrence Erlbaum Associates, Hillsdale.
Elmasri, R., y B. Shamkant. 2000. *Sistemas de Bases de Datos: conceptos fundamentales*. Addison Wesley Iberoamericana.
Ericsson, K., y H.A. Simon. 1993. *Protocol Analysis. Verbal reports as data*. (rev. ed). Cambridge Massachusets.: MIT Press.

Fauxx, Rob. 2006. Impact of pre-programing course curriculum on learning the first programming course. *IEEE Transactions on education* 49, nº. 1 (Febrero): 11-15.

Feinberg, Susan, y Margaret Murphy. 2000. Applying cognitive load theory to the design of web-based instruction. En *Proceedings of IEEE professional communication society international professional communication conference and Proceedings of the 18th annual ACM international conference on Computer documentation: technology & teamwork*, 353 - 360. Cambridge, Massachusetts: IEEE Educational Activities Department.

Felder, R. 1996. Matters of Style. *ASEE Prism*.

Fixx, Vikki, Susan Wiedenbeck, y Jean Scholtz. 1993. Mental Representations of Programs by Novices and Experts. En *Conference on Human Factors in Computing Systems*, 74-79. Proceedings of the SIGCHI conference on Human factors in computing systems. Amsterdam, The Netherlands: ACM Press New York, NY, USA.

Flavell, J.H. 1979. Metacognition and cognitive monitoring: A new area of cognitive developmental inquiry. *American Psychologist*.

Gagné, R.M. 1968. Learning Hierarchies. *Educational Psychologist* 6: 1-9.

Gane, C., y T. Sarson. 1977. *Structured Systems Analysis: Tools and Techniques*. McDonnell Douglas Systems Integration Company. http://portal.acm.org/citation.cfm?id=578678.

George, Carlisle E. 2000. Experiences with Novices: The Importance of Graphical Representations in Supporting Mental Models. En *12th Workshop of the Psychology of Programming Interest Group*. A.F. Blackwell & E. Bilotta.

Gerjets, Peter, Katharina Scheiter, y Richard Catrambone. 2004. Designing Instructional Examples to Reduce Intrinsic Cognitive Load: Molar versus Modular Presentation of Solution Procedures. *Instructional Science* 32, nº. 1: 33-58.

Gibbons, A.S. 2000. The Nature and Origin of Instructional Objects. En *The Instructional Use of Learning Objects.* Bloomington: Association for Educational Communications and Technology.

Gordon, V., y J. Bieman. 1995. Rapid Prototyping: lessons learned. *IEEE Software* 12, nº. 1: 85-95.

Gourgey, Annette F. 1998. Metacognition in basic skills instruction. *Instructional Science* 26, nº. 1: 81-96. doi:10.1023/A:1003092414893.

Guzdial, M.S. 2002. Teaching the Nintendo generation how to program. *Communications of the ACM* 45, nº. 4 (Abril): 17-21.

Hagan, Dianne, y Selby Markham. 2000. ¿Does it help to have some programming experience before beginning of a computer degree program? En *Annual Joint Conference Integrating Technology into Computer Science Education*, 25 - 28. Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education. Helsinki, Finland.

Hamer, J. 2004. A lightweight visualiser for Java. En *Proceedings of the Third Program Visualization Workshop*, 54-61. Warwick, UK.

Hegarty, M.J. 1993. Constructing mental models from text and diagrams. *J. Mem.*

Lang. 32: 717–742.
Heggestad, E.D.K. 2005. The Predictive Validity of Self-Efficacy in Training Performance: Little More Than Past Performance. *Journal of Experimental Psychology:* 11, nº. 2: 84-97.
Hevner, A., S. March, y J. Park. 2004. Design Science in Information Systems Research. *MIS Quarterly* 28, nº. 1 (Marzo): 75-105.
Holden, Edward, y Elisa Weeden. 2005. Prior Experience and New IT Students. *Issues in Informing Science and Information Technology* 2: 189.
Hu, C. 2004. Rethinking of Teaching Objects-First. *Education and Information Technologies* 9, nº. 3: 209-218.
Hu, Chenglie. 2006. It's Mathematical, after all -the nature of learning computer programming. *Educational Information Technology* 11: 83-92. doi:10.1007/s10639-005-5714-4.
Jehng, J. 1999. A visualisation approach to learning the concept of recursion. *Journal of Computer Assisted Learning* 15: 279-290.
Jenkins, T. 2002. On the difficulty of learning to program. En *3rd annual LTSN-ICS Conference*. Loughborough University: LTSN Centre of information and computer sciences.
Johnson-Laird, P.N. 1983. *Mental Models*. Cambridge University Press.
Jonassen, David H. 1997. Instructional design models for well-structured and III-structured problem-solving learning outcomes. *Educational Technology Research and Development* 45, nº. 1: 65-94. doi:10.1007/BF02299613.
Kalyuga, S. 2007a. Enhancing Instructional Efficiency of Interactive E-learning Environments: A Cognitive Load Perspective. *Educational Psychology Review* 19: 387–399. doi:10.1007/s10648-007-9051-6.
———. 2007b. Expertise Reversal Effect and Its Implications for Learner-Tailored Instruction. *Educational Psychology Review* 19, nº. 4: 509-539.
Kalyuga, S., P. Ayres, P. Chandler, y J. Sweller. 2001. The expertise reversal effect. *Educational Psychologist* 38: 23-32.
Kasper, G. 1996. A Theory of Decision support system design for user calibration. *Information Systems Research* 7, nº. 2: 215-232.
Kelleher, C.P., y Randy Pausch. 2005. Lowering the Barriers to Programming: a survey of programming environments and languages for novice programmers. *ACM Computing surveys (CSUR)* 37, nº. 2: 83 - 137.
Keppens, J., y D. Hay. 2008. Concept Map Assessment for Teaching Computer Programming. *Computer Science Education* 18, nº. 1: 31-42.
Knox, S.T., W.A. Bailey, y E.F. Lynch. 1989. Directed dialogue protocols: verbal data for user interface design. En *Proceedings of the SIGCHI conference on Human factors in computing systems: Wings for the mind*, 283 - 287. ACM New York, NY, USA.
Koedinger, K., V. Aleven, N. Hefferman, B. McLaren, y M. Hockenberry. 2004. Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration. En *Proceedings of 7th Annual Intelligent Tutoring Systems Conference*. Maceio, Brazil.
Koedinger, K., y J. Anderson. 1997. Intelligent Tutoring Goes To School in the Big City. *International Journal of Artificial Intelligence in Education* 8: 30-43.

Kolb, D.A. 1984. *Experiential learning*. Englewood Cliffs, NJ: Prentice Hall.
Kölling, M. 1999. Of teaching object-oriented programming, Part I: Languages. *Journal of Object-Oriented Programming* 11, nº. 8: 8-15.
Koong, Kai S., Lai C. Liu, y Xia Liu. 2002. A Study of the Demand for Information Technology
Professionals in Selected Internet Job Portals. *Journal of Information Systems Education* 13, nº. 1: 21-28.
Kounios, John, y Philip Holcomb. 1994. Concreteness effects in semantic processing: ERP evidence supporting dual-coding theory. *Journal of Experimental Psychology: Learning, Memory and Cognition* 20, nº. 4: 804-823.
Krahmer, Emiel, y Nicole Ummelen. 2004. Thinking About Thinking Aloud: A Comparison of Two Verbal Protocols for Usability Testing. *IEEE TRANSACTIONS ON PROFESSIONAL COMMUNICATION* 47, nº. 2: 105-117.
Kumar, Amruth N. 2006a. Using Enhanced Concept Map for Student Modeling in Programming Tutors. En *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*, 527-532. Melbourne Beach, Florida.
———. 2006b. The Effect of Using Problem-Solving Tutors on the Self-Confidence of Students. En , 275 - 283. University of Sussex.
Kuo, Mei-Ling Amy, y Simon Hooper. 2004. The Effects of Visual and Verbal Coding Mnemonics on Learning Chinese Characters in Computer-Based Instruction. *Educational Technology Research and Development* 52, nº. 3: 23-34.
Larkin, J., y H.A. Simon. 1987. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*.
LeFevre, J., y P. Dixon. 1986. Do written instructions need examples? *Cognition and Instruction* 3, nº. 1: 1-30.
Linn, M.C. 1985. The Cognitive Consequences of Programming Instruction in Classrooms. *Educational Researcher* 14.
Linn, M.C., y M.J. Clancy. 1992. The Case for Case Studies of Programming Problems. *Communications of the ACM* 35, nº. 3.
Litecky, Chuck, Bipin Prabhakatar, y Kirk Arnett. 2006. The IT/IS job market: a longitudinal perspective. En *Proceedings of the 2006 ACM SIGMIS CPR conference on computer personnel research: Forty four years of computer personnel research: achievements, challenges & the future*, 50-52. Claremont, California, USA.
Ma, Linxiao, John Ferguson, Marc Roper, y Murray Wood. 2007. Investigating the viability of mental models held by novice programmers. *ACM SIGCSE Bulletin* 39, nº. 1: 499-503.
Machanick, Philip. 2005. Peer Assessment for action learning of data structures and algorithms. En *Australasian Computing Education Conference*. Vol. 42. New Castle, Australia.
Mandel, T. 1997. *The elements of user interface design*. Wiley.
Maries, Adrian, y Amruth Kumar. 2007. Concept maps in intelligent tutors for

programming. *Journal of Computing Sciences in Colleges* 22, nº. 3: 54.
Markus, M. Lynne, Ann Majchrzak, y Less Gasser. 2002. A Design Theory for Systems That Support Emergent Knowledge Processes. *MIS Quarterly* 26, nº. 3: 179-212.
Mayer, R.E. 1992. *Thinking, problem solving, cognition*. New York: W.H. Freeman and Co.
———. 1998. Cognitive, metacognitive, and motivational aspects of problem solving. *Instructional Science* 26: 49–63.
Mayer, R.E., y Richard B. Anderson. 1991. Animations need narrations: An experimental test of a dual-coding hypothesis. *Journal of Educational Psychology* 83, nº. 4: 484-490.
Mayer, R.E., y M.C. Wittrock. 1996. Problem-solving transfer. En *Handbook of educational psychology*, 47-62. New York: Macmillan.
McGill, T.J., y S.E. Volet. 1997. A Conceptual Framework for Analyzing Students' Knowledge of Programming. *Journal of Research on Computing in Education* 29.
McIver, L. 2001. Syntactic and Semantic Issues in Introductory Programming Education. Monash University, AustraliaEditor.
McKeown, J. 2004. The use of a multimedia lesson to increase novice programmers' understanding of programming array concepts. *Journal of Computing Sciences in Colleges* Volume 19, nº. 4 (Abril): 39 - 50.
van Merrienboer, J. 1990. Strategies for programming instruction in high school: Program completion vs. program generation. *Educ. Comput. Res.* 6: 265-287.
van Merrienboer, J., y M. De Croock. 1992. Strategies for computer-based programming instruction: Program completion vs. program generation. *Journal of educational computing research*.
van Merrienboer, J., y H. Krammer. 1987. Instructional strategies and tactics for the design of introductory computer programming courses in high school. *Instructional Science* 16: 251-285.
———. 1990. The "completion strategy" in programming instruction: Theoretical and empirical support. En *Research on Instruction*, 45-61. Englewood Cliffs, NJ: Educational Technology Publications.
Meyer, Richard E., y Valerie K. Sims. 1994. For Whom Is a Picture Worth a Thousand Words? Extensions of a Dual-Coding Theory of Multimedia Learning. *Journal of Educational Psychology* 86, nº. 3: 389-401.
Miles, Don, Toni Blum, Wayne J. Staats, y David Dean. 2003. Experiences with the Metacognitive Skills Inventory. En *33rd Annual FIE*, T3B - 8-13 Vol.1. Boulder, CO. doi:10.1109/FIE.2003.1263324.
Milne, Iain, y Glenn Rowe. 2002. Difficulties in Learning and Teaching Programming—Views of Students and Tutors. *Education and Information Technologies* 7, nº. 1: 55-66.
Moisey, S.M. 2003. Fulfilling the promise of learning objects. En *In Handbook of Distance Education*, 1st ed.:323. Lawrence Erlbaum.
Moreno, Roxana. 2006. When worked examples don't work: Is cognitive load theory at an Impasse? *Learning and Instruction* 16: 170-81.

Moskal, Barb, Deborah Lurie, y Stephen Cooper. 2004. Evaluating the Effectiveness of a New Instructional Approach. En *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, 75 - 79. Norfolk, Virginia, USA.

Neisser, Ulrich. 1967. *Cognitive psychology*. 1º ed. Englewood Cliffs, NJ, USA: Prentice Hall.

Newell, A., y H.A. Simon. 1972. *Human Problem Solving*. Englewood Cliffs, NJ: Prentice Hall.

Nielsen, J. 1993. *Usability Engineering*. Cambridge, Massachusetts: Academic.

Nielsen, J., T. Clemmensen, y Y. Carsten. 2002. Getting access to what goes on in people's heads?: reflections on the think-aloud technique. En *Proceedings of the second Nordic conference on Human-computer interaction*, 31:101 - 110. Aarhus, Denmark: ACM New York, NY, USA.

Nielsen, Jakob. 1993. *Usability Engineering*. San Diego, CA: Academic press.

Norman, D.A. 1983. *Some observations on mental models.* Erlbaum, Hillsdale, NJ.

Novak, J., y A. Cañas. 2006. *La Teoría Subyacente a los Mapas Conceptuales y Cómo Construirlos*. Reporte Técnico. Reporte Técnico IHMC CmapTools 2006-01. Florida: Institute for Human and Machine Cognition (IHMC).

Novak, J. D. 1998. *Learning, creating, and using knowledge: Concept Maps(R) as facilitative tools in schools and corporations*. Mahweh, NJ: Lawrence Erlbaum Associates.

Novak, Joseph D., y Alberto J. Cañas. 2006. The Origins of the Concept Mapping Tool and the Continuing Evolution of the Tool. *Information Visualization Journal* 5, nº. 3: 175-184.

Orlikowsky, W., y C. Iacono. 2001. Research Commentary: Desperately Seeking the “IT” in IT
Research – A Call to Theorizing the IT Artifact. *Information Systems Research* 12, nº. 2: 121-134.

Ousterhout, J. 1998. Scrpting: higher level programming for the 21st century. *IEEE Computer* 31, nº. 3: 23-30.

Oviatt, Sharon. 2006. Human-centered design meets cognitive load theory: designing interfaces that help people think. En *Proceedings of the 14th annual ACM international conference on Multimedia*, 871 - 880. Santa Barbara, CA, USA: ACM New York, NY, USA.

Paivio, Allan. 1990. *Mental Representations: a dual coding approach*. New York: Oxford University Press.

Papert, S. 1980. *Mindstorms, children, computers and powerful ideas*. Basic Books, New York.

Phalp, K., y S. Counsell. 2001. Coupling Trends in Industrial Prototyping Roles: An Empirical Investigation. *Software Quality Journal* 9: 223–240.

Pollack, S.B.-A. 2004. Selecting a Visualization System. En *Third Program Visualization Workshop*. Warwick, UK.

Pont, M. 1998. Why Java is dangerous. *IEEE Software*.

Pressman, Roger S. 2002. *Ingenieria de Software, un enfoque práctico*. 5º ed. Madrid, España: McGraw-Hill.

de Raadt, Michael, Richard Watson, y Mark Toleman. 2003. Language tug-of-war:

industry demand and academic choice. En *Proceedings of the fifth Australasian conference on Computing education*, 137 - 142. Adelaide, Australia.

Ramalingam, V.L. 2004. Self-Efficacy and mental models in learning to program. En *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, 171 - 175. Leeds, United Kingdom: ACM Press New York, NY, USA.

Renkl, Alexander, Robert K. Atkinson, y Cornelia S. Große. 2004. How Fading Worked Solution Steps Works – A Cognitive Load Perspective. *Instructional Science* 32, nº. 1: 59-82.

Renkl, Alexander, Tatjana Hilbert, y Silke Schworm. 2009. Example-Based Learning in Heuristic Domains: A Cognitive Load Theory Account. *Educational Psychology Review* 21, nº. 1: 67-78.

Rist, R.S. 1989. Schema Creation in Programming. *Cognitive Science* 13: 389-414.

———. 1995. Program Structure and Design. *Cognitive Science* 19, nº. 4: 507-562.

———. 1996. System Structure and Design. En *Empirical studies of programmers: sicth workshop*. Norwood, New Jersey: Ablex Publishing Corporation.

———. 2004. Learning to program: schema creation, application and evaluation. En *Computer Science Education and Research*, 175-197. Netherlands: Taylor & Francis Group.

Rob, P., C. Coronel, y K. Crocket. 2008. *Database systems: design implementation and management*. Carnegie Learning EMEA. http://books.google.com.mx/books?hl=es&lr=&id=gCzfjlWOVAAC&oi=fnd&pg=PA2&dq=%09+Database+systems:+a+practical+approach+to+design,+implementation,+and+management&ots=By4KxVYoMA&sig=rogMLja9R9OpLrCfuwLfhFKif-o#v=onepage&q=Database%20systems%3A%20a%20practical%20approach%20to%20design%2C%20implementation%2C%20and%20management&f=false.

Robins, A., J. Rountree, y N. Rountree. 2003. Learning and Teaching Programming: A Review. *Computer Science Education* 13, nº. 2: 137–172.

Roll, Ido, Vincent Aleven, Bruce McLaren, y Keneth Koedinger. 2007. Designing for metacognition—applying cognitive tutor principles to the tutoring of help seeking. *Metacognition Learning* 2: 125–140. doi:10.1007/s11409-007-9010-0.

Rößling, G.N. 2002. A Testbed for Pedagogical Requirements in Algorithm Visualizations. *ACM SIGCSE Bulletin* 34, nº. 3: 96 - 100.

Roth, Wolff-Michael. 2004. Theory and praxis of metacognition. *Pragmatics & Cognition* 12, nº. 1: 153–168.

Rumbaugh, J., I. Jacobson, y G. Booch. 2000. *El lenguaje unificado de modelado (UML). Manual de referencia*. Madrid: Addison Wesley. Pearson Education.

———. 2004. *The rational unified process: an introduction*. Object Technologies. Boston, MA. USA.: Addison Wesley. http://books.google.com/books?id=RYCMx6o47pMC&pg=PR2&dq=RUP,+rumbaugh,+jacobson&hl=es&cd=5#v=onepage&q=RUP%2C%20rumbaugh%

2C%20jacobson&f=false.
Russo, J.E., E.J. Jonson, y D.L. Stephens. 1989. The validity of verbal methods. *Memory Cognition* 17, nº. 6: 759-769.
Sadoski, Mark, y Allan Paivio. 2004. A Dual Coding Theoretical Model of Reading. En *Theoretical models and processes of reading*. 5º ed. Newark, DE: International Reading Association.
Sajaniemi, J.H. 2006. Teaching Programming: Going beyond “Objects First". En *18th Workshop of the Psychology of Programming Interest Group,* University of Sussex.
Schoenfeld, A.H. 1987. What’s all the fuss about metacognition? En *Cognitive science and mathematics education*. Hillsdale N.J.: Lawrence Erlbaum Associates.
Simon, Sally Fincher, Antony Robins, y Bob Baker. 2006. Predictors of Success in a First Programming Course. En *Eighth Australasian Computing Education Conference (ACE2006)*. Vol. 52. Hobart, Tasmania, Australia.
Simon, H.A. 1996. *The sciences of the artificial*. MIT Press.
Smith, S. M., T.B. Ward, y I.S. Schumacher. 1993. Constraining effects of examples in a creative generation task. *Memory Cognition* 21: 837-845.
Soloway, E., y K. Ehrlich. 1984. Empirical studies of programming knowledge. *IEEE Transactions on software engineering* 10: 595-609.
Sommerville, Ian. 2002. *Ingeniería de Software*. 6º ed. Mexico: Pearson Education.
Spiro, R., Brian P. Collins, Jose J. Thota, y Paul J. Feltovitch. 2003. Cognitive Flexibility Theory: Hypermedia for Complex Learning, Adaptive Knowledge Application, and Experience Acceleration. *Educational Technology* 43, nº. 5: 5-10.
Spohrer, J.C., y E. Soloway. 1989. Novice mistakes: Are the folks wisdoms correct? En *Studying the novice programmer*, 401-416. New Jersey: Lawrence Erlbaum Associates, Hillsdale.
Stein, F., y V. Zwass. 1995. Actualizing Organizational Memory with Information Systems. *Information Systems Research* 6, nº. 2: 85-117.
Sweller, J. 1988a. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science* 12: 257-285.
———. 1988b. Cognitive load during problem solving: Effects on learning. *Cognitive Science* 12: 257-285.
Sweller, J., y G. Cooper. 1985. The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction* 2: 59- 89.
Sweller, J., J. van Merrienboer, y F. Paas. 1998. Cognitive Architecture and Instructional Design. *Educational Psychology Review* 10, nº. 3: 251-295.
Sweller, John. 1994. Cognitive Load Theory, Learning Difficulty and Instructional Design. *Learning and Instruction* 4: 295-312.
Tanik, M.M., y R. T. Yeh. 1989. Rapid Prototyping in Software Development. *IEEE Computer* 22, nº. 5: 9-11.
Thomas, Lynda, Mark Ratcliffe, John Woodbury, y Emma Jarman. 2002. Learning Styles and Performance in the introductory programming sequence. *ACM SIGCSE Bulletin* 34, nº. 1 (Marzo): 33 - 37.
Trafton, J.G., y B.J. Reiser. 1993. The Contribution of Studying Examples and

Solving
Problems to Skill Acquisition. En *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, 1017-1022. Erlbaum, Hillsdale, NJ.

UAA, Departamento de Estadistica Institucional. 2007. *Porcentajes de reprobación en la materia de Programación I, en las carreras de LI-LTI/ISC durante los años 2004 y 2007*. Datos Internos. Aguascalientes: Universidad Autónoma de Aguascalientes.

Urquiza-Fuentes, J., y J.A. Velázquez-Iturbide. 2009. A Survey of Successful Evaluations of Program Visualization and Algorithm Animation Systems. *ACM Transactions on Computing Education (TOCE). Special Issue on the 5th Program Visualization Workshop (PVW'08)* 9, nº. 2.

VanLehn, K. 1996. Cognitive skill acquisition. *Annual Review of Psychology* 47: 513-539.

Volet, S.E., y C.P. Lund. 1994. Metacognitive instruction in introductory computer programming: A better explanatory construct for performance than traditional factors. *Journal of educational computing research* 10.

Von Mayrhauser, A., y A.M. Vans. 1994. *Program understanding – A survey*. Técnico. Colorado, EU: Department of Computer Science, Colorado State University.

Walls, Joseph G., George R. Widmeyer, y Omar El Sawy. 1992. Building an Information System Design Theory for Vigilant EIS. *Information Systems Research* 3, nº. 1: 36-59.

———. 2004. Assesing information system design theory in perspective: how useful was our 1992 initial rendition? *Journal of information technology theory and application* 6, nº. 2: 43-58.

Ward, M., y J. Sweller. 1990. Structuring effective worked examples. *Cognition and Instruction* 7, nº. 1: 1-39.

Weber, R. 1987. Toward a Theory of Artifacts: A Paradigmatic Base for Information Systems Research. *Journal of Information Systems* 1, nº. 2: 3-19.

Wells, T. 2006. Usability: reconciling theory and practice. En *Proceedings of the 24th annual ACM international conference on Design of communication*, 99 - 104. Myrtle Beach, SC, USA.

White, Garry. 2003. Standarized mathematics scores as a prerequisite for a first programming course. *Mathematics and Computer Education* 37, nº. 1.

Wiedenbeck, Susan, Deborah LaBelle, y Vennila Kain. 2004. Factors affecting course outcomes in introductory programming. En *16th Workshop of the psychology of programming interest group*. Institute of Technology, Carlow, Ireland.

Wiley, D. 2000. *The Instructional Use of Learning Objects*. Bloomington, IN: Association for Educational Communications and Technology.

Williams, Laurie, y Richard L. Upchurch. 2001. In support of student pair-programming. *ACM SIGCSE Bulletin* 33, nº. 1: 327 - 331.

Winslow, L.E. 1996. Programming Pedagogy - A psychological Overview. *SIGCSE Bulletin* 28: 17-22.

Yourdon, E. 1989. *Modern Structured Analysis*. Prentice Hall.

Zhu, X, y H.A. Simon. 1987. Learning mathematics from examples and by doing. *Cognition and Instruction* 4: 137-166.