SEP TNM

INSTITUTO TECNOLÓGICO DE CULIACÁN

Ambiente de Aprendizaje de Depuración de Errores de Programación Basado en Ejemplos Erróneos

TESIS

PRESENTADA ANTE EL DEPARTAMENTO ACADÉMICO DE ESTUDIOS DE POSGRADO DEL INSTITUTO TECNOLÓGICO DE CULIACÁN EN CUMPLIMIENTO PARCIAL DE LOS REQUISITOS PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

POR:

Elvia Minerva Valencia Rodríguez
LICENCIADA EN INFORMÁTICA

DIRECTOR DE TESIS:

Dra. María Lucía Barrón Estrada

CULIACÁN, SINALOA NOVIEMBRE 2017

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Culiacán

"Año del Centenario de la Promulgación de la Constitución Política de los Estados Unidos Mexicanos"

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Culiacán, Sin., 17 de Noviembre del 2017

OFICIO: DEPI-538/XI/2017
ASUNTO: **Autorización Impresión**

LIC. ELVIA MINERVA VALENCIA RODRÍGUEZ
ESTUDIANTE DE LA MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN
PRESENTE.

Por medio de la presente y en virtud de que ha completado los requisitos para el examen de grado de la **Maestría en Ciencias de la Computación**, se concede autorización para la impresión de la tesis titulada: **" AMBIENTE DE APRENDIZAJE DE DEPURACIÓN DE ERRORES DE PROGRAMACIÓN BASADO EN EJEMPLOS ERRÓNEOS"**, bajo la dirección del(a) **Dra. María Lucía Barrón Estrada**

Sin otro particular reciba un cordial saludo.

ATENTAMENTE
"CON LA TÉCNICA AL PROGRESO"

SEP TecNM
Instituto Tecnológico de Culiacán
División de Estudios de Posgrado e Investigación

M.C. MARÍA ARACELY MARTÍNEZ AMAYA
JEFE(A) DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

C.c.p. archivo
DNU/lucy *

" AMBIENTE DE APRENDIZAJE DE DEPURACIÓN DE ERRORES DE PROGRAMACIÓN BASADO EN EJEMPLOS ERRÓNEOS"

Tesis presentada por:

LIC. ELVIA MINERVA VALENCIA RODRÍGUEZ

Aprobada en contenido y estilo por:

Dra. María Lucía Barrón Estrada
Director de Tesis

Dr. Ramón Zatarain Cabada
Secretario

M.C. Gloria Ekaterine Peralta Peñuñuri
Vocal -1

M.C. Rosalío Zatarain Cabada
Vocal -2

M.C. María Aracely Martínez Amaya
Jefe(a) de la División de Estudios de Posgrado, Investigación

Dedicatoria

A Dios por darme vida y fuerza para continuar cada día.

A mis maestros, porque confiaron en mí y me dedicaron tiempo y esfuerzo.

A mi familia y amigos, por su interés y apoyo en que logre la culminación del presente trabajo de tesis.

A maestros y estudiantes avanzados en ciencias de la computación que deseen realizar mejoras en el área de educación en ciencias de la computación.

Agradecimientos

Son muchas las personas y condiciones que contribuyeron para que se pudiera realizar el presente trabajo de tesis, mismas que me permito mencionar y agradecer:

A Dios, por hacerme sentir acompañada en todo momento, por la fe que me consuela, me anima y fortalece cada día.

A mis padres Anastacio Valencia Rosales y María Dolores Rodríguez Garma, por haberme dado sus vidas y no solo darme la mía, por su infinito amor y apoyo incondicional.

A mis hermanos, Talía Hebe, Laura Leticia, Alicia Dolores y Jorge Armando, por su afecto, alegría y vivencias compartidas que aligeran la carga, a la que a veces confronta la vida.

A mi esposo Ramón, por su gran apoyo y paciencia, por sus consejos y protección que me han permitido sentirme plena y agradecida, por ser el mejor amigo, esposo y padre.

A mis hijos Sara Daniela, Ramón Santiago y Ana Sofía, por su amor incondicional que me fortalece cada mañana, por su entusiasmo por la vida y su desesperación de querer aprender todo lo que van descubriendo, que me sorprende y motiva.

A mi directora de tesis María Lucía Barrón, por su ayuda durante toda la maestría y el desarrollo del presente trabajo de tesis, por haberme dado la oportunidad, herramientas y apoyo para aprender y crecer.

A todos los maestros que me impartieron clase durante la maestría, Ramón Zatarain, María Lucía Barrón, Ricardo Rafael Quintero, Emir Abel Manjarrez y Carlos Santillán, por haberme ofrecido el conocimiento de una manera única y excelente, por su experiencia y consejos.

A la maestra Gloria Ekaterine Peralta y al maestro Jesús Astolfo Rodríguez, por haberme permitido realizar con sus alumnos los experimentos del presente trabajo de tesis asignando el tiempo necesario para el desarrollo las actividades, lo cual agradezco y valoro profundamente.

A mis amigos de generación de maestría, Bianca Giovanna Medina, Ana Cecilia Lara, Fernando Quintero, Daniel Leal, Christian Jair Lindor y Noel Andrés García, que siempre

llevaré en mi mente y corazón por haber compartido conmigo no solo sus conocimientos, sino también experiencias de vida, por haberme dado ánimos cuando lo necesité, por haber compartido juntos estrés, desvelos, risas y diversión.

A Francisco González, Raúl Oramas y Mario Ríos, por haberme asesorado cuando lo necesité, por sus consejos y su amistad.

A Conacyt y al Instituto Tecnológico de Culiacán, por el apoyo económico y alta calidad de enseñanza que me ofrecieron y que me deja con una gran experiencia educativa y enorme gratitud.

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Elvia Minerva Valencia Rodríguez.

Culiacán, Sinaloa, México, 2017

Resumen

El uso de las computadoras ha permeado en todas las áreas del quehacer humano y hoy en día existen muchas personas interesadas en el desarrollo de software que satisfaga algunas necesidades, pero desarrollar programas no es una tarea trivial y requiere habilidades que deben desarrollarse para lograr programas eficientes y libres de errores.

La tarea de programación implica diversas etapas y la depuración de errores es una que se dificulta particularmente a los estudiantes con poca experiencia en desarrollo de programas.

Este trabajo de tesis presenta un ambiente de aprendizaje llamado *Find Error Java*, cuyo objetivo es ofrecer a los programadores novatos tutoría sobre depuración de errores de programación en lenguaje Java.

Para el desarrollo del ambiente de aprendizaje se consideró mejorar el conocimiento cognitivo y metacognitivo de los estudiantes, con un enfoque de enseñanza basada en ejemplos erróneos. El diseño de la instrucción se basó en los errores comunes de programación identificados en una población de estudiantes del Instituto Tecnológico de Culiacán. Enfocados a mejorar las características de *Find Error Java*, su arquitectura fue diseñada considerando los componentes básicos de un Sistema Tutor Inteligente, utilizando la técnica pedagógica cognitiva *example-tracing*, además se agregaron elementos de gamificación e intervenciones del agente pedagógico Lucy (Sosa Ochoa, 2016) con el objetivo de influir en la motivación del estudiante.

Todo lo anteriormente descrito se integró en un ambiente web disponible para cualquier computadora con acceso a internet que utilice cualquier sistema operativo Linux, Windows o Mac OSX.

Se efectuó un estudio utilizando la herramienta con estudiantes con el objetivo de medir la usabilidad (mediante evaluación TAM) y la efectividad en ganancias de aprendizaje (mediante la aplicación de exámenes Pre-Test y Pos-Test) y los resultados obtenidos fueron muy alentadores.

Palabras clave

Programación Java

Depuración de errores

Ejemplos erróneos

Example-Tracing

Programadores novatos

Ambientes de aprendizaje

Agente pedagógico

Gamificación

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

1. Introducción

La educación en ciencias de la computación específicamente, la enseñanza-aprendizaje de la programación ha recibido un notable interés por parte de la comunidad científica prácticamente desde sus inicios, esto se debe, principalmente a la complejidad que representa enseñar y aprender programación (Jenkins, 2002).

Cualquier persona que aprenda a programar debe enfrentarse a la tarea de corregir sus propios errores, lo cual, para los programadores novatos es algo desafiante incluso para aquellos que tienen una buena comprensión de la sintaxis (Ahmadzadeh, Elliman, & Higgins, 2005). Una deficiente habilidad en depuración de errores produce frustración en el estudiante y además puede ocasionar la introducción de nuevos errores (Murphy et al., 2008).

Los Sistemas Tutores Inteligentes (STI) y los Ambientes de Aprendizaje Inteligentes aportan un gran apoyo a los cursos de programación tradicionales (instructor frente a grupo) debido a que se adaptan al estudiante y proporcionan tutoría personalizada que sería difícil impartir debido a la cantidad de estudiantes en un salón de clases. Sin embargo la gran mayoría de estos STI, ofrecen poca ayuda en el sentido de generar habilidades en depuración de errores ya que su enfoque principal es enseñar nuevos conceptos.

En la mayoría de las investigaciones relacionadas con la instrucción en ciencias de la computación, se han utilizado métodos de enseñanza basados en ejemplos correctos y mínimamente se han considerado a los ejemplos erróneos, tal vez por el miedo a que los estudiantes asimilen los errores y los repitan, sin embargo, diversos estudios en diferentes áreas del conocimiento tales como Matemáticas (Borasi, 1994), Medicina (Kopp, Stark, & Fischer, 2008) e incluso Computación (Ginat & Shmalo, 2013) han reportado evidencia que demuestra que el uso de los errores para la instrucción aporta un beneficio significativo en la adquisición de conocimiento.

En este trabajo de investigación se presenta un ambiente de aprendizaje de depuración de errores de programación basado en ejemplos erróneos llamado *Find Error Java,* el cual tiene como objetivo que los estudiantes mejoren sus habilidades de depuración de errores mientras

refuerzan conceptos y principios necesarios para la comprensión de dichos errores. Durante la realización de los ejercicios el estudiante realiza un razonamiento consciente sobre las fases del proceso de depuración que debe efectuar para corregir los errores, las cuales van desde el entendimiento de la notificación del error (instrucciones del ejercicio), pasando por la identificación del error en el código (línea o fragmento de código erróneo), entendimiento de la causa (conceptos, reglas o principios relacionados) y corrección del error (instrucciones correctas).

Para el diseño de la instrucción se consideraron principios cognitivos, así como también aspectos constructivistas en el sentido que los ejercicios consideran conocimiento previo (pudiendo éste ser impreciso). Se utilizaron ejemplos erróneos ya que, el conflicto cognitivo que aportan los ejemplos erróneos ha sido considerado por la psicología constructivista como un catalizador para el aprendizaje en el que se deben realizar revisiones de conocimientos y procedimientos (Confrey, 1990).

Aunado a lo anterior, se integró a la estrategia de instrucción basada en ejemplos erróneos, al agente pedagógico Lucy (Sosa Ochoa, 2016) cuyo objetivo fue generar empatía con el estudiante ofreciéndole pistas y retroalimentación inmediata a nivel de pasos. Además, cuando Lucy detecta que el estudiante está fallando repetidamente, realiza intervenciones con el objetivo de motivarlo a que realice reflexiones más profundas antes de intentar responder a la pregunta planteada. Siguiendo con el tema de la motivación, se utilizó una mecánica de gamificación la cual consiste en la obtención de puntos (estrellas) cada vez que el estudiante contesta correctamente y se le regala un punto extra (una estrella más) si contesta correctamente en el primer intento. El sistema incluye también un tablero de los puntajes de todos los participantes para incentivar su interés en comprender los temas. Para ganar puntos extra, se utilizó un enfoque de recompensa y no de penalización con la intención de generar un ambiente relajado y de diversión.

El capítulo se divide de la siguiente manera: la sección 1.1 corresponde al planteamiento del problema, en la 1.2 se presenta la justificación, la sección 1.3 realiza el planteamiento de la hipótesis, la 1.4 presenta el objetivo general, la 1.5 describe los objetivos específicos y finalmente la sección 1.6 muestra la estructura de la tesis.

1.1. Planteamiento del problema

La depuración de errores está implícita en la actividad de programar, tanto el programador experto como el novato deben realizar este proceso en algún momento. Para los programadores novatos es lógico que esta actividad sea en mayor medida desafiante y frustrante.

A pesar de que en los cursos de programación, libros de texto e investigaciones existentes se reconoce la importancia de la actividad de depuración de errores, pocas veces es considerada como tema central en la enseñanza. Por lo tanto, la adquisición de la habilidad para depurar programas generalmente se deja a cargo del mismo estudiante, encontrándose solo durante el proceso de adquisición de este conocimiento.

1.2. Justificación

Las clases tradicionales de diversas áreas del conocimiento generalmente usan ejemplos correctos, debido tal vez a su utilidad para desarrollar un tema. Se ha observado que los estudiantes a partir de los ejemplos correctos crean asociaciones, de las cuales unas son correctas pero otras son vagas o incorrectas (Vinner, 1983), por lo que es importante aplicar estrategias que confronten al estudiante con esas inferencias erróneas.

Durante la codificación de un programa pueden suceder descuidos, conocimientos faltantes o conceptos erróneos que provoquen una gran variedad de errores, la depuración de errores es el reflejo de todos los faltantes cognitivos, de ahí la complejidad de esta tarea ya que además de lo anterior los estudiantes principiantes carecen de estrategias de depuración que les ayuden a identificar y resolver los errores. Los errores del estudiante ofrecen una gran oportunidad para aprender de ellos.

El uso de ejemplos erróneos ha sido utilizado en Matemáticas, Física y Medicina, entre otras áreas, aportando buenos resultados, sin embargo en Ciencias de la Computación la depuración de errores que es la analogía de ejemplos erróneos pocas veces o nunca es utilizada por los maestros, libros de programación y software educativo como estrategia central de enseñanza.

El presente proyecto aporta un enfoque y herramienta de software que ayuda a los estudiantes principiantes a desarrollar la habilidad de encontrar y eliminar errores en programas Java mientras adquieren o refuerzan conceptos profundos de una manera efectiva y sencilla.

1.3. Hipótesis

Con la implementación de prácticas de depuración de errores comunes en un ambiente de aprendizaje que aplique una estrategia de instrucción mediante técnicas de seguimiento de ejemplos (*exampe-tracing*) combinadas con el uso de un agente pedagógico y elementos de gamificación, se logrará que los estudiantes aprendan de una manera sencilla y ágil a evitar o superar fácilmente dichos errores así como también se pretende lograr que los estudiantes escriban programas completos con mayor dominio.

1.4. Objetivo general

Desarrollar un ambiente de aprendizaje de depuración de errores cuya estrategia de instrucción esté basada en ejemplos erróneos y la técnica de seguimiento de ejemplos (*example-tracing*) aplicada mediante un agente pedagógico y además el ambiente incluya elementos de gamificación que incentiven conductas adecuadas en el estudiante con el fin de mejorar su aprendizaje.

1.5. Objetivos específicos

Para lograr el objetivo general, se desarrollaron una serie de objetivos específicos que permiten descomponer el problema para facilitar su solución. A continuación se presenta la lista de objetivos específicos.

- Identificar los errores comunes que cometen los estudiantes novatos al aprender a programar.
- Crear un banco de ejercicios de ejemplos de programación bajo un enfoque constructivista.

- Diseñar e implementar la instrucción de depuración de errores mediante la adaptación a la técnica de seguimiento de ejemplos (*example-tracing*) y la consideración de principios cognitivos.
- Seleccionar, implementar e integrar dinámicas y mecánicas de gamificación que ayuden a motivar a los estudiantes y propicien la reflexión, en un ambiente lúdico.
- Integrar al agente pedagógico Lucy (Sosa Ochoa, 2016) la estrategia de instrucción basada en ejemplos erróneos.
- Diseñar e implementar el plan de pruebas experimentales para evaluar la usabilidad del sistema y la ganancia de aprendizaje de los estudiantes.

1.6. Estructura de tesis

En este documento se presentan las diferentes fases del proceso de investigación y desarrollo del tema de tesis Ambiente de Aprendizaje de Depuración de Errores de Programación Basado en Ejemplos Erróneos llamado '*Find Error Java'*.

El capítulo 2 presenta el marco teórico que le da sustento al trabajo. En él, se presenta la literatura relacionada con el presente tema de tesis, los conceptos, enfoques, métodos y técnicas consideradas. El capítulo 3 exhibe el estado del arte donde se presentan los diferentes sistemas que tienen relación con el sistema desarrollado en esta tesis, por tratarse de software educativo enfocado a la enseñanza de un lenguaje de programación, preferentemente Java u orientado a objetos. El capítulo 4 corresponde al desarrollo del proyecto, en este capítulo se presenta un panorama general del proceso de investigación aplicado y se detalla el proceso de desarrollo de software. El capítulo 5 detalla el método aplicado para la realización de los experimentos y los resultados obtenidos. En el capítulo 6 se discuten las conclusiones y trabajo futuro.

Capítulo 2

2. Marco teórico

En este capítulo se presenta la base teórica utilizada para la realización de este trabajo de tesis que aborda el desarrollo de una herramienta para la enseñanza del proceso de depuración de errores.

El capítulo se divide de la siguiente manera: En la sección 2.1 se aborda el tema de depuración de errores, en la sección 2.2 se explica el enfoque de ejemplos erróneos, la sección 2.3 corresponde al tema de tecnología para la educación tales como Sistemas Tutores Inteligentes y Ambientes de Aprendizaje, la sección 2.4 presenta el tema de gamificación y la 2.5 trata el tema de agentes pedagógicos.

2.1. Depuración de errores

La depuración de errores es una fase del ciclo de vida del software, es el proceso en el que se corrigen defectos que impiden el correcto funcionamiento del programa. Para los programadores novatos es una actividad desafiante incluso para aquellos que tienen una buena comprensión de la sintaxis (Ahmadzadeh et al., 2005), en ocasiones produciendo frustración y la introducción de nuevos errores (Murphy et al., 2008).

Se ha observado que los buenos programadores no necesariamente son buenos depuradores de errores, en cambio los buenos depuradores suelen ser buenos programadores (Fitzgerald et al., 2008) (Ahmadzadeh et al., 2005). Lo anterior fortalece la idea que la actividad de depuración refuerza en el programador sus habilidades y conocimiento del lenguaje.

Existen investigaciones que abordan el tema de la depuración de errores de programación desde diferentes enfoques tales como identificar el tipo de errores que comúnmente cometen los programadores novatos (Fitzgerald et al., 2008), la creación de patrones de comportamiento de depuración (Ahmadzadeh et al., 2005), mejoras a ambientes de desarrollo para visualización dinámica de programas (Cross, Hendrix, & Barowski, 2011), pronósticos de navegación en la web para la depuración de programas con el objetivo de ampliar

herramientas que ofrecen los entornos de desarrollo (Lawrance et al., 2013); por otra parte, también se han abordado factores afectivos como frustración ante la depuración (Murphy et al., 2008).

Algo importante de resaltar, es que la mayoría de estas investigaciones considera a la depuración de errores como una actividad de recuperación, posterior a la codificación y no como un tema central para la enseñanza.

2.1.1. Estrategias propuestas para la depuración de errores

Aunque existe una amplia variedad de estudios dedicados a la enseñanza de habilidades de depuración no ha habido un consenso en cuanto cuáles son dichas habilidades, ni cuáles son las estrategias apropiadas para su enseñanza.

Benander y Benander (Benander & Benander, 1989) sugieren que la enseñanza de la depuración consiste en enseñar técnicas tales como la inserción de instrucciones adicionales de salida en el código para producir salidas intermedias o usar el modo de ejecución paso a paso para darle seguimiento a la ejecución del programa. Por su parte Gugerty y Olson (Gugerty & Olson, 1986) en su estudio concluyeron que la comprensión del programa es el único factor que permite una depuración eficiente.

Ahmadzadeh y otros (Ahmadzadeh et al., 2005) proponen que la comprensión de patrones de depuración en errores de compilación y lógicos ofrece elementos que permiten mejorar el método de enseñanza, mientras que Murphy y otros (Murphy et al., 2008) plantean que las habilidades de depuración consisten en aplicar simultáneamente la comprensión del funcionamiento previsto del programa, la ejecución real defectuosa del programa, tener experiencia en programación en general, comprensión del lenguaje de programación, comprensión del dominio de la aplicación y tener conocimiento de errores y métodos de depuración; por ejemplo, el seguimiento de código, declaraciones de impresión de diagnóstico y la coincidencia de patrones.

En la literatura se proponen una amplia variedad de estrategias de depuración de errores, durante la revisión de estas estrategias se observaron dos enfoques generales:

- Estrategias cuyo objetivo es mejorar los procesos cognitivos; por ejemplo, el

reconocimiento de conceptos difíciles o conceptos erróneos y patrones.

- Estrategias cuyo objetivo es el desarrollo de un modelo mental de una máquina nocional; por ejemplo, la depuración paso a paso o la revisión dinámica de los valores de una variable. El concepto de máquina nocional (Boulay, O'Shea, & Monk, 1981), representa un modelo mental de un paradigma de programación específico que permite describir y predecir con precisión el comportamiento de un programa a medida que se ejecuta.

El alcance del presente trabajo de tesis incluye el primer enfoque mencionado, es decir, el objetivo principal es mejorar el proceso cognitivo del estudiante lo cual desde la hipótesis planteada mejorará sus habilidades de depuración de errores.

2.1.2. Tipos y causas de errores

Los errores pueden clasificarse en tres categorías generales, que son:

a. **Errores de sintaxis**. Son aquellos que infringen en las reglas gramaticales del lenguaje, por ejemplo el orden de las palabras que conforman una instrucción. Los errores de sintaxis hacen que el programa no sea comprendido por el compilador, generando mensajes de error durante la compilación, todos estos errores deben corregirse para que se pueda realizar la compilación.
b. **Errores semánticos**: Son aquellos que infringen la semántica del lenguaje, es decir, el significado del código. Para ejemplificar, un error semántico puede darse a partir de una idea equivocada de la manera en la que el lenguaje procesa las instrucciones. Estos errores se producen en un nivel más abstracto que los errores de sintaxis, puede ser que un código esté correcto en sintaxis sin embargo su semántica sea incorrecta. El compilador de java detecta algunos de estos errores durante la compilación, sin embargo otros pudieran causar una terminación inesperada de la ejecución del programa.
c. **Errores lógicos**. Son errores que permiten la compilación y la ejecución del programa sin embargo producen resultados incorrectos.

Esta clasificación ha sido útil en la impartición de cursos de programación, sin embargo, se ha percibido generalizadamente la necesidad de una identificación y clasificación más específica de estos errores, sobre todo en relación a los programadores novatos.

La investigación sobre los tipos de errores que cometen los programadores novatos y la manera en la que éstos deben ser considerados en la enseñanza sigue siendo, hasta la fecha, un tema abierto de investigación.

Jackson, Cobb y Carver (Jackson, Cobb, & Carver, 2005) en su trabajo de investigación, cuyo objetivo central es identificar los errores comunes que comenten los programadores novatos en lenguaje Java, mencionan que durante su revisión a la literatura encontraron que los métodos más usados para la identificación de errores comunes de programación son el uso de la experiencia, la aplicación de encuestas a maestros, el conteo manual y categorización de errores o bien, el uso de las compilaciones proporcionadas por los mismos alumnos. Ellos utilizaron este último para desarrollar un sistema automatizado de recolección de errores en tiempo real para programación Java, cuyo uso estaba dirigido a una población específica de estudiantes. El estudio dio como resultado la identificación de diez errores comunes, los cuales se presentan en la Tabla 2-1.

Tabla 2-1. Errores comunes obtenidos en estudio de Jackson y colegas.

Prioridad	Errores identificados
1	cannot resolve symbol
2	; expected
3	illegal start of expression
4	class or interface expected
5	<identifier> expected
6	) expected
7	incompatible types
8	int
9	not a statement
10	} expected

McCall y Kölling (McCall & Kölling, 2015) proponen un método de categorización de errores comunes cometidos por programadores novatos al utilizar el lenguaje de programación Java. El método propuesto por su estudio toma como base a los mensajes de diagnóstico producidos por el compilador. Mediante una herramienta web desarrollada para el estudio, dos investigadores categorizaron todos los errores, éstos podían ver el mensaje de error, el código que generó el error y contaban con una opción para categorizarlo; otro investigador realizó un refinamiento de la categorización y finalmente se aplicó un método de validación de la objetividad de la categorización. Las diez más frecuentes categorías y su correspondiente frecuencia obtenidas por su estudio son presentadas en la Tabla 2-2.

Tabla 2-2. Diez categorías más frecuentes de errores obtenidos en estudio de McCall y Kölling.

Prioridad	Errores identificados
11.1%	Variable not declared
10.3%	; missing
8.4%	Variable name written incorrectly
7.9%	Invalid Syntax
4.9%	Method name written incorrectly
4.1%	Missing parentheses for constructor call
3.0%	Unhandled exception
2.7%	Class name written incorrectly
2.4%	Method call: parameter type mismatch
2.4%	Type mismatch in assignment

Por otra parte, Hristova, Misra, Rutter y Mercuri (Hristova, Misra, Rutter, & Mercuri, 2003) identificaron errores comunes de programadores novatos, los cuales fueron utilizados en la construcción de un entorno de desarrollo de programación para el lenguaje Java llamado *Expresso*, el cual mejoraba los mensajes de compilación para facilitarle su comprensión al programador novato.

Clasificar los errores comunes de los programadores puede ser el punto de partida para la mejora de los métodos pedagógicos (con o sin uso de tecnología educativa), o bien, para la mejora de las características de los entornos de desarrollo para principiantes.

En la revisión realizada a la literatura, los esfuerzos de categorizar los errores comunes,

partiendo de los mensajes de error de compilación y ejecución, generalmente fueron motivados para la mejora de mensajes de compilación de entornos de desarrollo.

En el presente trabajo de tesis, no solo se realizó un análisis de los mensajes de error de compilación y ejecución para conocer los errores frecuentes de una población específica de estudiantes, también se revisaron exámenes de evaluación del curso, observación naturalista durante prácticas de laboratorio y entrevistas a maestros y estudiantes de programación.

Por otro lado, también se han estudiado las causas de los errores de programación.

Du Boulay (Du Boulay, 1986) propuso que el uso incorrecto de analogías de un tema a otro o la generalización excesiva son las causas de los errores de los programadores novatos. Un ejemplo de esto es como los estudiantes aplican su entendimiento de un procedimiento del paradigma orientado a objetos con la invocación de un método de un objeto sin la indicación de dicho objeto.

Por otra parte, Rath y Brown (Rath & Brown, 1995) consideraron que algunos estudiantes tienen la idea de la existencia de un "razonamiento independiente de la computadora" o "que la computadora se encargará de eso". Los autores mencionan que la situación más común que ejemplifica dicho pensamiento es cuando el programador novato determina que su algoritmo está correctamente codificado porque es claro para él y por lo tanto la computadora lo entenderá. Otro ejemplo de esta simplificación exagerada, o bien, pensamiento de que la computadora se encargará, es cuando los estudiantes, ante una declaración de un objeto piensan que la computadora se encargará de la creación del objeto. Este último ejemplo de error también se presentó durante nuestras observaciones naturalistas de prácticas de laboratorio realizadas para el presente trabajo de tesis. Otros investigadores también han observado errores de estudiantes que tenían el supuesto que la computadora realizará algunas acciones sin que hayan sido especificadas (Sleeman, Putnam, Baxter, & Kuspa, 1986) .

Sleeman y otros (Sleeman et al., 1986) concluyeron que muchos errores se producen por una confusión entre el lenguaje de programación y el lenguaje natural.

Este fenómeno respecto a la confusión entre el lenguaje de programación y el lenguaje natural también se presentó durante nuestras observaciones naturalistas de prácticas de laboratorio desarrolladas para el presente trabajo de tesis, en donde por ejemplo, un estudiante en un

programa utilizó una variable llamada *sumatoria* y en otra línea de código se refirió a ésta como *sumatotal*, el estudiante no lograba encontrar el error, y cuando finalmente se le informó el motivo del error, le tomó un momento comprender que eran variables diferentes, el lenguaje natural había tomado una importancia por encima de lo que ya conocía del manejo de las variables. Los esfuerzos por categorizar los errores y sus causas sigue siendo un tema de investigación abierto.

2.2. Ejemplos erróneos

La exposición a errores produce conflicto cognitivo, en el que se provoca la revisión de los conocimientos (cognición) y procedimientos (metacognición) (Confrey, 1990). Con mayor detalle es posible decir que los ejemplos erróneos, en el aspecto cognitivo generan la necesidad de revisión de los conocimientos previos o bien generan actividad mental (no siempre consciente) necesaria para procesar la información y darle un sentido significativo. Además, los ejemplos erróneos, en el aspecto metacognitivo provocan un aprendizaje mediante la conciencia sobre la manera en la que se está aprendiendo.

En el área de las matemáticas es donde mayormente se han utilizado a los errores para la enseñanza. Borasi (Borasi, 1994) argumenta que se debe tomar ventaja de los errores, viendo a éstos como oportunidades de aprendizaje en la enseñanza. Sus estudios han demostrado que es posible obtener beneficios si se utilizan adecuadamente. Propone que se debe exponer al estudiante a errores no solo con el objetivo de corregirlos, sino de realizar una exploración y reflexión acerca de éstos, además sostiene que el uso adecuado de los errores conduce al estudiante incluso a su erradicación.

Para el área de ciencias de la computación, Ginat y Shmalo (Ginat & Shmalo, 2013), desarrollaron un enfoque basado en errores para un curso introductorio a la programación orientada a objetos. El enfoque se basa en un conjunto de principios los cuales son:

a) conflicto cognitivo, el cual se genera cuando el estudiante se da cuenta que el resultado obtenido no es lo que esperaba;
b) atribución de errores, el cual consiste en la ejecución de un proceso de “asignación de culpa”, es decir, la búsqueda de la causa del error;

c) actividad en clase, puede ser individual, en parejas y/o grupal, el objetivo es buscar una solución, que eventualmente producirá una revisión del conocimiento;
d) explicación, que se refiere a una autoexplicación tanto del error como de la solución;
e) reflexión, es una discusión en clase que concluye con una reflexión sobre el proceso de solución, se discuten los conceptos erróneos.

El alcance establecido en este estudio se enmarca en la comprensión de las características del lenguaje y no incluye la resolución de problemas.

Existen estudios que han combinado ejemplos erróneos con tecnología educativa. En matemáticas, (Melis, 2004) (Melis, Sander, & Tsovaltzi, 2010) creó un entorno de aprendizaje llamado *ActiveMath,* en sus experimentos se pudo observar una contribución cognitiva (matemáticas) y metacognitiva (conciencia del proceso de análisis). En medicina Kopp, Stark y Fischer (Kopp et al., 2008) crearon un entorno de aprendizaje para facilitar conocimiento de diagnóstico a estudiantes. Por otra parte, Yoon, Kang y Kwon (Yoon, Kang, & Kwon, 2014), en el área de las ciencias de la computación, crearon *DeBugger*, cuyo objetivo es promover el aprendizaje en depuración de errores de programación.

En diversas investigaciones se ha demostrado que el uso adecuado de ejemplos erróneos conduce a buenos resultados, aplicarlos en un entorno tecnológico educativo incluye, entre otros aspectos, el diseño de la instrucción, que se refiere al proceso o método de enseñanza, tal como la definición de las habilidades que deben ser fortalecidas, la decisión sobre la manera y el momento para dar retroalimentación al estudiante y la elección o innovación de los recursos pedagógicos-tecnológicos tales como el uso de agentes pedagógicos o recursos de gamificación. El campo de investigación en pro de la mejora del proceso de enseñanza es fértil.

2.3. Tecnología para la educación

Las computadoras se han utilizado desde hace muchos años en la educación creándose el área denominada Aprendizaje Electrónico (eLearning en inglés). Zhang y otros (Zhang, Zhao, Zhou, & Nunamaker, 2004) definen e-Learning como el aprendizaje basado en tecnología donde el material de aprendizaje se entrega electrónicamente a los estudiantes utilizando una

red de computadoras, que puede ser abierta (Internet) o cerrada como una (intranet).

El aprendizaje electrónico tiene ventajas sobre los sistemas tradicionales que requieren que el profesor y los estudiantes se encuentren en un mismo lugar en un tiempo determinado; pero, los sistemas electrónicos usados en la enseñanza deben combinar diferentes elementos para que puedan actuar como lo haría un tutor real.

Existen una gran variedad de aplicaciones que pueden utilizarse para e-Learning, el tipo de aplicación puede variar, puede ser una simulación, un entorno de aprendizaje abierto, un juego, un sistema de realidad virtual o una colaboración grupal. Por otro lado, los recursos científicos sobre el entendimiento de como las personas aprenden (psicología y ciencia cognitiva), o las investigaciones sobre el aprendizaje máquina (inteligencia artificial), así como los avances tecnológicos (computadoras, internet, telecomunicaciones), pueden ser utilizados en conjunto para la creación de éstas aplicaciones (Woolf, 2010).

2.3.1. Principios cognitivos para el diseño de instrucción

El diseño de instrucción se refiere a la definición de las modalidades educativas, los criterios de organización de la información y la estrategia de enseñanza que permite generar experiencias de aprendizaje significativas. El diseño de la instrucción es una fase fundamental en la construcción de un Sistema Tutor Inteligente (STI).

A continuación se mencionan algunos de los principios cognitivos útiles para el diseño de instrucción:

a) Conocimiento previo, se refiere al conocimiento que ya posee el estudiante antes de dar inicio a la instrucción. En cuanto al enfoque de ejemplos erróneos, se ha demostrado que el beneficio de su uso depende de la relación entre el nivel de conocimiento del alumno y la dificultad del ejemplo (Melis et al., 2010). Por tanto los ejemplos deben corresponder al nivel de conocimiento del estudiante.
b) Andamiaje, se refiere al proceso que proporciona estructuras de apoyo que permiten gradualmente la adquisición de nuevos conocimientos.

c) Carga cognitiva, se le denomina así a la idea de que la memoria de largo plazo tiene una gran capacidad de almacenamiento de esquemas (patrones de conocimiento), en cambio, la memoria de trabajo tiene una capacidad limitada por lo que es necesario dosificar la cantidad de conocimiento nuevo.
d) Intercalación, sugiere que durante una lección se ofrezca al estudiante una mezcla de tipos de problemas en lugar de un solo tipo.
e) Retroalimentación, es la explicación que realiza el estudiante acerca del error, la cual no debe limitarse a solo la indicación de la corrección. La retroalimentación promueve una comprensión profunda de los temas, conceptos y procesos cognitivos involucrados. Puede realizarse en lenguaje natural mediante texto escrito por el estudiante, o bien, mediante opción múltiple. En esta última forma, su utilidad radica en que es posible darle información que le servirá para complementar su entendimiento, por ejemplo, si al estudiante se le da la opción de elegir entre una lista el tipo de error que se le presentó, se le estará ayudando a demás a generar un modelo mental de categorías de errores.

2.3.2. Sistemas Tutores Inteligentes

No existe una definición universal aceptada de Sistema Tutor Inteligente. Puede decirse que los Sistemas Tutores Inteligentes (STI) son aquellos que ofrecen una enseñanza diferencial, aquellos que adaptan su respuesta de enseñanza después de realizar un razonamiento sobre las necesidades de los estudiante y el conocimiento del dominio (Woolf, 2010).

La mayoría de los autores coinciden en la distribución típica de la arquitectura y los componentes principales que la integran (Sottilare, Graesser, Hu, & Goldberg, 2014) siendo estos el modelo del dominio, modelo del estudiante, modelo del tutor e interfaz de usuario. A continuación se presenta una breve descripción de cada uno de estos componentes.

- **Modelo del dominio**: El dominio se refiere al tema del cual se va a dar tutoría, como por ejemplo el dominio de las matemáticas, física, química o programación. El modelo del dominio, posee el conocimiento experto del dominio, es decir, contiene las definiciones, procesos, estrategias, habilidades así como también errores y conceptos erróneos típicos.

- **Modelo del estudiante**: Puede ser conceptualizado como un subconjunto del modelo del dominio, es decir, mediante este componente, es posible conocer el estado cognitivo del estudiante (conocimiento adquirido y faltante) al realizar una superposición de éste con el modelo del dominio. Además de representar el estado cognitivo también pudiera incluir el estilo de aprendizaje, o los estados afectivos y de motivación del estudiante. Este componente permite al módulo tutor comprender el estado en el que se encuentra el estudiante, de tal forma que el modelo del tutor pueda adaptar las experiencias de tutoría a las necesidades específicas del estudiante.
- **Modelo del tutor**: Este componente tiene la responsabilidad de comprender el estado en el que se encuentra el estudiante con respecto al modelo del dominio, es decir, tiene la responsabilidad de comprender qué conocimientos le falta por adquirir o reforzar al estudiante. Es a este componente al que se le puede asignar la responsabilidad de determinar la estrategia de instrucción y de intervenciones considerando por ejemplo el estilo de aprendizaje, el estado afectivo o de motivación del estudiante. En resumen, este componente determina la estrategia de instrucción una vez analizado el modelo de dominio y modelo del estudiante. Este componente también es llamado, modelo pedagógico o modelo de instrucción.
- **Interfaz de usuario**: Este componente permite interacciones humano-computadora tales como las que tradicionalmente se establecen mediante teclado o *mouse,* e interacciones más complejas, por ejemplo las que se establecen utilizando lenguaje natural (mediante texto escrito o por voz) o bien, el reconocimiento de emociones del estudiante mediante lectura de señales electro-encefalográfícas (EEG).

En la Figura 2-1 se presenta la arquitectura comúnmente aceptada de un STI.

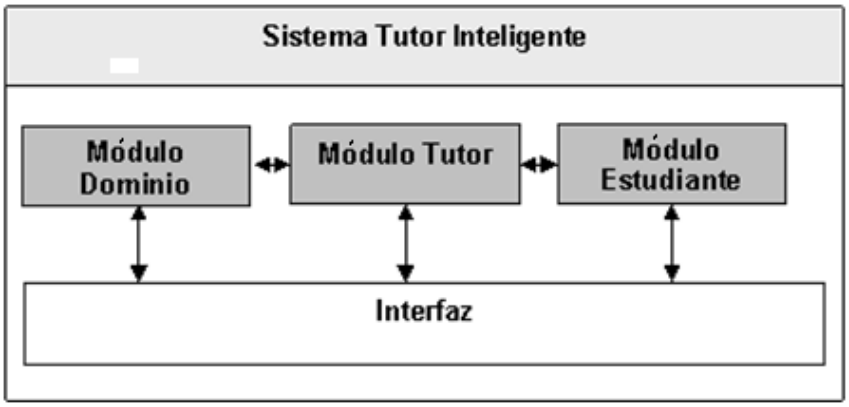


Figura 2-1. Distribución típica de componentes de Arquitectura de un STI.

2.3.3. Metodologías para la representación del conocimiento experto

Las metodologías que se explicarán a continuación tienen un enfoque cognitivo, el cual se sustenta en la identificación de los procesos mentales durante el aprendizaje y en principios como los mencionados en el punto 2.3.1

a) ***Model-Tracing*:** Este método realiza una interpretación del comportamiento del estudiante mediante el uso de reglas generalizadas (reglas de producción) basadas en un modelo cognitivo de resolución experta de problemas (Aleven et al., 2016).
b) **Restricciones:** Este método permite realizar una interpretación y evaluación del trabajo del estudiante con respecto a un conjunto de restricciones que deben cumplierse satisfactoriamente en su totalidad (Mitrovic & Ohlsson, 1999). Este método opera bajo la idea de que es suficiente detectar que el estudiante ha cometido un error.
c) ***Example-Tracing*:** A diferencia del método *model-tracing* o del método basado en restricciones, este método evalúa el comportamiento del estudiante comparándolo flexiblemente contra ejemplos generalizados de comportamiento de resolución de problemas. (Aleven, Mclaren, Sewall, & Koedinger, 2009).

En *example-tracing* los ejemplos generalizados, contienen las rutas de solución aceptables para un problema en particular (ruta óptima y sub-óptimas) además, también es posible registrar comportamiento erróneo (correspondiente a errores comunes) para

ofrecer tutoría contextualizada a un tipo de error en particular y tienden a ser más fáciles de crear que las reglas de producción o las restricciones.

Los tutores basados en el método *example-tracing* pueden exhibir un comportamiento sofisticado de tutoría, ya que es posible proporcionar orientación a nivel paso, es decir, orientación durante la resolución de problemas complejos y no solo orientación hasta la finalización del ejercicio lo que hace que se cumpla con el criterio mínimo para considerase como un STI según (Vanlehn, 2006).

Aleven y otros (Aleven et al., 2009), describen el proceso de autoría de un tutor basado en *example-tracing*, utilizando una herramienta de autor llamada CTAT, la cual no requiere que el autor tenga conocimientos de programación. A continuación se presentan los aspectos de autoría que para el presente trabajo de tesis son relevantes:

1. El autor debe estudiar el pensamiento y el aprendizaje del dominio mediante la técnica de análisis de tareas cognitivas (Clark, Feldon, van Merriënboer, Yates, & Early, 2008), también puede aplicar métodos de minería de datos educacionales.
2. El autor procede a diseñar y desarrollar una o más interfaces de usuario mediante las cuales se dará tutoría a un estudiante generalmente para un tipo de problema en particular, donde un problema debe ser descompuesto en pasos de solución.
3. El autor crea grafos de comportamiento generalizado, donde las aristas representan los pasos en la solución de problemas en la interfaz y los nodos representan los estados. El grafo debe capturar las diferentes rutas (o formas) de solución del problema, de tal manera que el tutor pueda evaluar las acciones del usuario contra las del grafo. Es importante reiterar que en el grafo se especifica la solución óptima y las sub-óptimas y de ser necesario también los errores que comúnmente se cometen. La generalización del grafo de comportamiento, consiste en indicar un rango de comportamiento válido para un paso determinado, es decir, se generan rutas de solución donde un mismo paso es correcto aunque con diferentes valores, utilizando fórmulas por ejemplo.

4. Se deben adjuntar consejos (*hints*) a las aristas del grafo, los cuales serán utilizados durante la tutoría, también es importante adjuntar mensajes de retroalimentación (*feedback*) en respuesta a un error específico.

CTAT, es una herramienta de autor mediante la cual es posible crear tutores *example-tracing* sin necesidad de tener conocimientos de programación. En la Figura 2-2, Figura 2-3 y Figura 2-4 se presenta una secuencia que ilustra la manera en la que se genera un grafo de comportamiento desde esta herramienta para la resolución de un problema de fracciones.

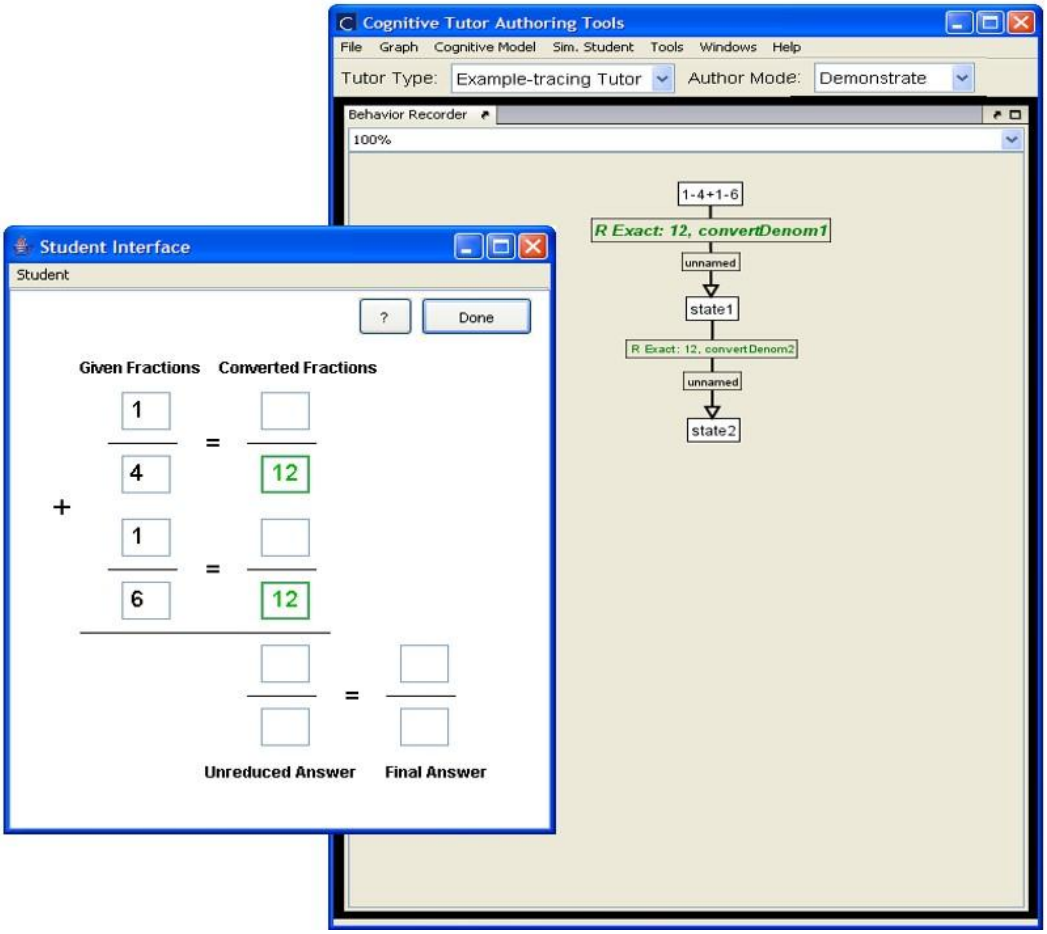


Figura 2-2. Configuración de grafo de comportamiento de los primero dos estados.

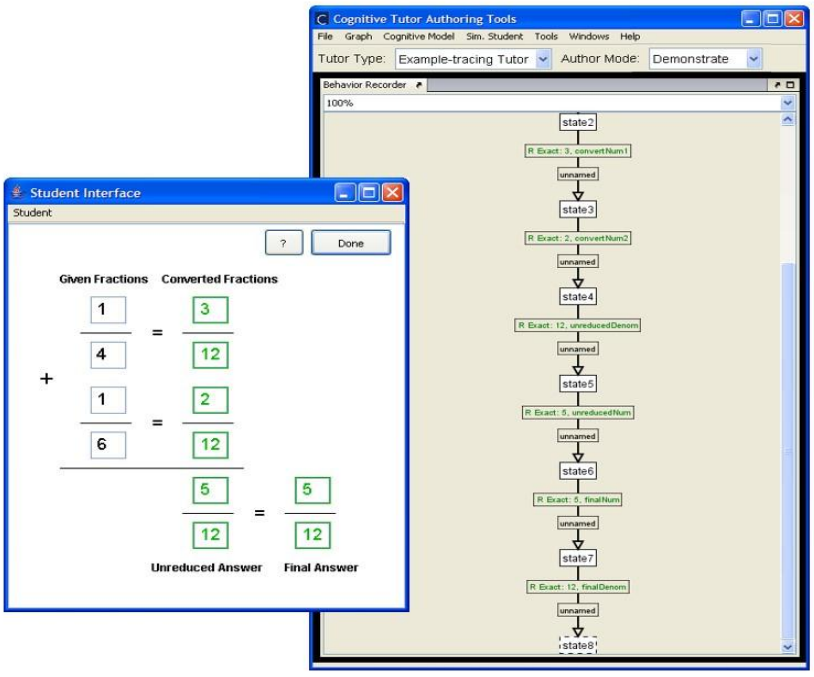


Figura 2-3. Continuación de creación de primera ruta de solución del grafo de comportamiento.

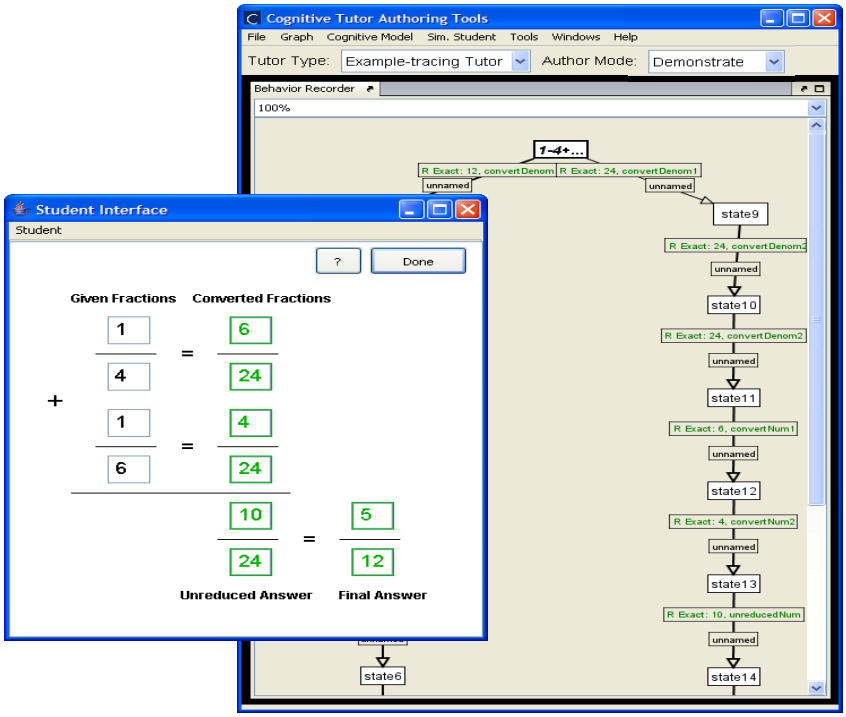


Figura 2-4. Adición de ruta alterna de solución al grafo de comportamiento.

En la Figura 2-5 se presenta un ejemplo de cómo se configura en CTAT un mensaje de retroalimentación para un tipo de error en particular y en la Figura 2-6 se presenta la imagen de la interfaz de usuario mostrando el mensaje de error.

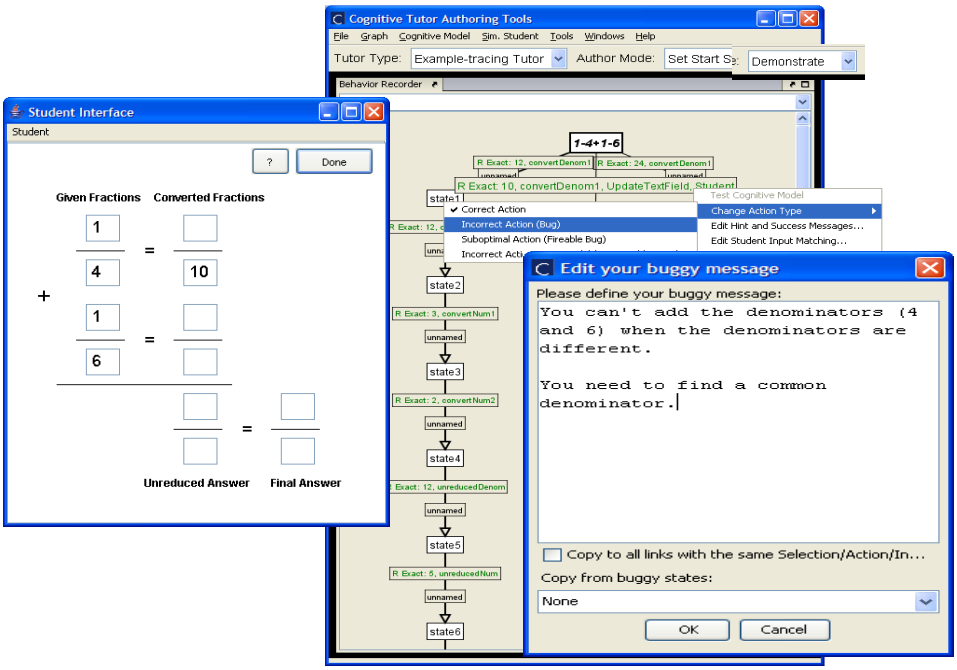


Figura 2-5. Configuración de mensaje de retroalimentación ante un error específico.

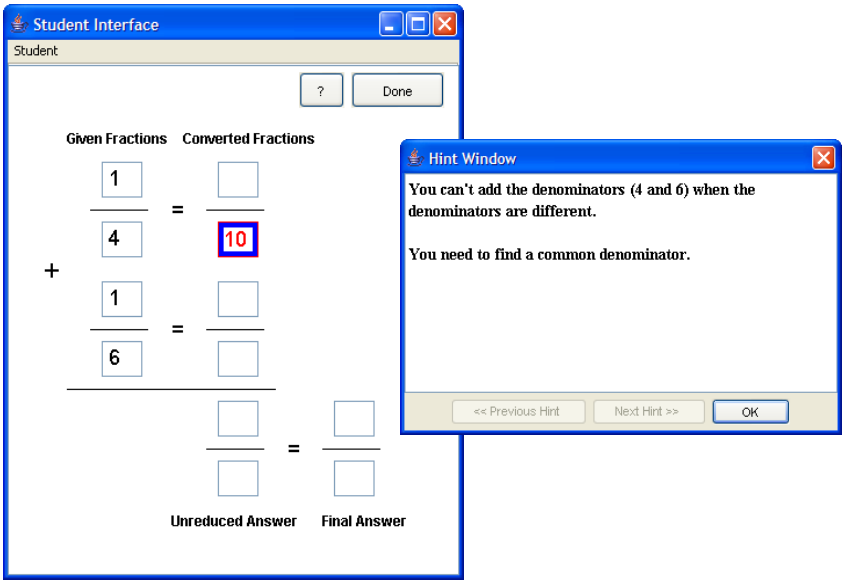


Figura 2-6. Interfaz de usuario de tutor exhibiendo mensaje de retroalimentación.

2.3.4. Ambientes de aprendizaje

Los STI se caracterizan por poseer mecanismos de inteligencia artificial que permiten adaptar la tutoría, pero generalmente no brindan al estudiante la posibilidad de elegir los temas o ejercicios del dominio. Los ambientes de aprendizaje a diferencia de los STI, ponen a disposición del estudiante todo el contenido educativo ofreciéndole la posibilidad de elegir los temas y secuencia de sus actividades de estudio, sin embargo, no cuentan con características de inteligencia artificial que permita guiar la tutoría.

Los Entornos Inteligentes de Aprendizaje (EIA) incorporan características de los ambientes de aprendizaje (flexibilidad de navegabilidad en los temas de instrucción) y los Sistemas Tutores Inteligentes (adaptabilidad de las experiencias de tutoría conforme al modelo del estudiante). Estas características que anteriormente eran consideradas como contradictorias hoy se consideran como complementarias (Brusilovsky, Pesin, & Zyryanov, 1993).

En el presente trabajo de tesis se construyó un Ambiente de Aprendizaje, sobre una arquitectura diseñada para soportar requerimientos adaptativos correspondientes a un Entorno Inteligente de Aprendizaje.

2.4. Gamificación

Los videojuegos se han vuelto muy populares en los últimos años propiciando una industria prolífica que genera millones de dólares en ganancias[1]. La popularidad de los videojuegos ha motivado a los investigadores de otras áreas a utilizar algunos de los elementos presentes en los videojuegos, en los sistemas aplicados en contextos diferentes al entretenimiento como por ejemplo la educación; a esto se le conoce como uso de gamificación en el aprendizaje.

Aunque el término gamificación, surgió en el ámbito digital y tuvo una aceptación generalizada después del 2010 (Deterding, Dixon, Khaled, & Nacke, 2011), es importante mencionar que desde los años ochenta los investigadores han estudiado los beneficios de los enfoques basados en juegos en la educación (Malone & Lepper, 1987)(Gee, 2003).

[1] Factbox: A look at the $65 billion video games industry. June 6, 2011. Reuters. http://uk.reuters.com/article/2011/06/06/us-videogames-factbox-idUKTRE75552I20110606.

En el presente trabajo de tesis el uso del término gamificación se refiere a la inclusión de elementos (mecánicas y dinámicas) presentes en los videojuegos, en el diseño y desarrollo de aplicaciones correspondientes a un contexto educativo, con la intención de involucrar, motivar y mejorar el aprendizaje. Los elementos de gamificación no deben ser aplicados para convertirse en el tema central de una aplicación sino como una característica cuyo propósito es motivar al estudiante a utilizarla (Raymer, 2011).

Las mecánicas de juego se refieren a las reglas que se deben cumplir para producir cambios en el sistema, generalmente se trata de recompensas virtuales (Chorney, 2012), como por ejemplo ganar medallas, puntos, insignias o monedas, o bien, subir de nivel o categoría, así como visualizaciones de tablero de jugadores, entre otras.

Las dinámicas de juego se refieren a los efectos que las mecánicas tienen sobre la experiencia del usuario. En el estudio de (Stott & Neustaedter, 2013) hablan sobre las dinámicas subyacentes que hacen que los juegos sean atractivos y cómo éstas han sido utilizadas en prácticas pedagógicas modernas, aunque se llaman de diferente manera, mencionan por ejemplo, que el concepto de libertad para fallar usado en videojuegos se relaciona con el concepto de evaluación formativa, en la cual se realizan evaluaciones y retroalimentación continua sin afectar en la calificación del estudiante. Otro ejemplo es la progresión de diseño usada en el contexto de juego que se relaciona con el aprendizaje escalonado o con uso de andamios.

Mencionan que hay una característica que comparten las dinámicas de juego exitosas, la cual se refiere a provocar una sensación de *agency* y *ownership,* que se puede interpretar en la sensación del usuario (jugador/estudiante) de ser el autor mismo de sus propias acciones y haberlas hecho por decisión propia, es decir, en un contexto de videojuegos, el jugador hace que sucedan eventos porque así quiso o pudo hacerlo, esa misma sensación debe ser fomentada en el contexto educativo.

2.5. Agentes pedagógicos.

Uno de los elementos importantes en los sistemas educativos es el tutor o profesor que determina la estrategia de enseñanza a utilizar para que los estudiantes adquieran el

conocimiento. En los sistemas usados en el aprendizaje electrónico, los tutores son llamados agentes pedagógicos (AP) y frecuentemente se representan a través de personajes que pueden ser reales o animados; ellos se encargan de establecer la comunicación con los estudiantes interactuando de diferentes formas para proporcionarles el apoyo cognitivo que requieren.

En la Figura 2-7 se muestran un ejemplo de un agente pedagógico realizta y otro animado.

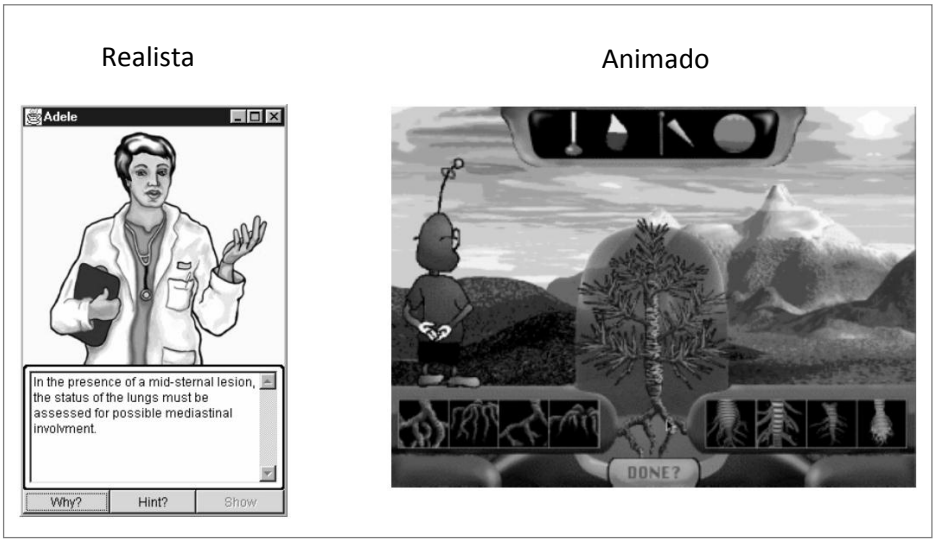


Figura 2-7. Agente pedagógico realista y animado

Los AP tienen una característica única, la cual radica en su capacidad para agregar un componente social al entorno de aprendizaje (Baylor & Kim, 2005).

Los AP actualmente pueden mostrar características humanas en su interacción; por ejemplo: utilizan diversas expresiones faciales para representar emociones (Zatarain Cabada, Barron Estrada, Gonzalez Hernandez, & Oramas Bustillos, 2015), movimientos del cuerpo (Wiggins et al., 2015) o diálogos en lenguaje natural (Kopp, Gesellensetter, Krämer, & Wachsmuth, 2005). Pueden ser implementados en un entorno de aprendizaje, representando un rol específico, pudiendo fungir el rol de tutor experto (Graesser, Person, & Harter, 2001), mentor (Baylor, 2000), compañero de aprendizaje (Chan & Chou, 1997) o colaborador (Dillenbourg & Self, 1992). Baylor y Kim (Baylor & Kim, 2005) consideran tres roles como los principales:

- **Experto**: Demuestra maestría o amplio conocimiento así como un rendimiento mejor que el promedio dentro de un dominio, impone autoridad.
- **Motivador**: Ofrece estímulo verbal, apoya y motiva al estudiante a realizar tareas difíciles, muestra comportamientos expresivos y maneja un lenguaje entusiasta y común.
- **Mentor**: Proporciona una guía para el aprendiz, no maneja una figura autoritaria, sin embargo refleja experiencia y conocimineto avanzado al mismo tiempo que motiva al alumno, muestra comportamiento expresivo, amigable, accesible pero profesional.

Capítulo 3

3. Estado del arte

Muchos trabajos de investigación que abordan la enseñanza de la programación, perciben al proceso de depuración de errores como una actividad de recuperación, que solo tiene lugar después de la escritura del código. La depuración de errores como tema central, se presenta en muy pocos trabajos de investigación a pesar de que se han observado beneficios substanciales en el estudiante cuando se utiliza como una herramienta pedagógica.

En este capítulo se presenta una revisión de los trabajos de investigación relacionados al aprendizaje de la programación, así como aquellos que abordan la enseñanza de la depuración de errores como tema central, ambos tipos de software dirigidos al tipo de estudiante denominado en la literatura como novato o principiante.

3.1. iSnap

iSnap (Price, 2017) es una extensión del entorno de programación para novatos llamado Snap el cual a su vez está basado en Scratch. La extensión iSnap realiza el registro detallado de todas las acciones de los estudiantes en una base de datos remota y ofrece sugerencias de siguiente paso bajo demanda. Estas sugerencias se generan automáticamente utilizando el algoritmo de descomposición de árbol contextual (CTD, por sus siglas en inglés), el cual se utiliza para realizar una comparación entre el código actual y las soluciones previas de los estudiantes.

Las sugerencias basadas en datos como las que utiliza iSnap no requieren de expertos para construir un modelo de contenido de dominio, por lo tanto, se considera que son fáciles de generar. Sin embargo este tipo de sugerencias basadas en datos tienen una limitante, la cual radica en que, ante la ausencia de un experto, el estudiante solo recibe sugerencias respecto a qué debe hacer, pero no recibe una explicación fundamental del por qué debe seguir el consejo.

En la Figura 3-1 se puede observar el entorno de programación de iSnap donde aparece un icono (parte izquierda derecha) que puede usar el estudiante para solicitar una sugerencia.

Figura 3-1. Estudiante selecciona burbuja de sugerencia.

En la Figura 3-2 se presenta el entorno de programación de iSnap donde el estudiante recibe la sugerencia de código. La interfaz permite que el estudiante evalúe la sugerencia de código recibida así como también solicitar otras sugerencias de código. El código sugerido no viene acompañado de una explicación que permita al estudiante entender por qué recibió esa sugerencia en particular.

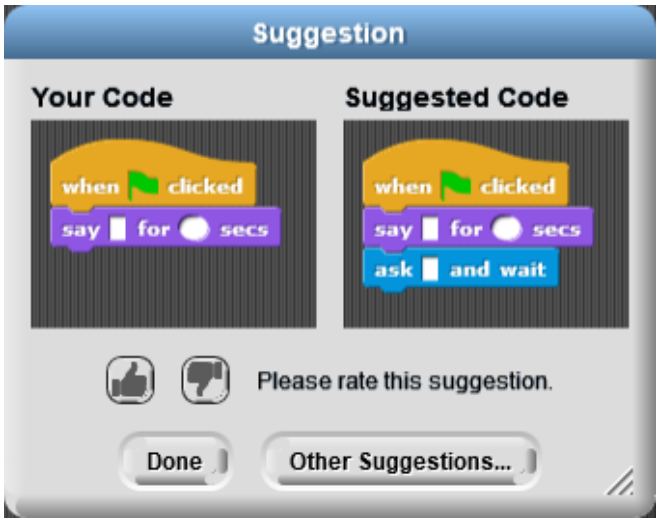


Figura 3-2. Sugerencia de código.

3.2. Java Assist Learning System (JALs)

Java Assist Learning System (JALs) (Lai, Lin, Jong, & Hsia, 2013) es una aplicación creada bajo el enfoque colaborativo de aprendizaje entre pares. Ésta utiliza una API que Facebook proporciona para desarrollar programas. Los estudiantes, dentro de la red social cuentan con una herramienta para realizar la codificación y compilación de programas Java. Cuando un estudiante requiere apoyo para poder realizar correctamente su código, mediante un mecanismo interno, Java Assist Learning System, busca a otro estudiante que anteriormente

haya resuelto el problema y establece un diálogo entre ellos, solo en caso que no encuentre a otro estudiante, buscará a un experto el cual aparentará ser otro estudiante más. Esta característica permite que los estudiantes puedan aclarar sus dudas en tiempo real estableciendo comunicación con otros estudiantes que resolvieron exitosamente el problema.

3.3. JavaTutor

JavaTutor (Wiggins et al., 2015) es un Sistema Tutor Afectivo (STA) multimodal que enseña programación en Java. Se fundamenta en la detección de estados afectivos del estudiante con el fin de adaptarse a los usuarios. La detección del estado emocional del estudiante se efectúa a través de distintos dispositivos de hardware que permiten realizar un reconocimiento de la expresión facial, gestos a distancia y postura corporal. El sistema, en algunas tareas específicas establece un diálogo con el estudiante en formato de *chat*, cuyo objetivo es que el estudiante verbalice un plan para resolver la tarea asignada.

En la Figura 3-3 se presenta la interfaz de usuario del Sistema Tutor Afectivo (STA) JavaTutor. La interfaz de JavaTutor contiene diferentes secciones; al lado izquierdo de la ventana se aprecian tres partes principales: en la parte de arriba se describe la tarea (*task*) que el estudiante debe realizar, en el centro se encuentra una sección para que el estudiante escriba el código de programación correspondiente con sus botones para compilar y ejecutar el programa, finalmente en la sección de abajo se muestran los mensajes que recibe el estudiante de parte del compilador e intérprete de Java. En la parte derecha de la interfaz, se encuentran dos secciones: en la sección de arriba se presenta la ventana donde el sistema presenta pistas al estudiante; en la parte de abajo, se presenta la información sobre los temas de estudio que se están explorando en esta tarea.

Figura 3-3. Interfaz de usuario de JavaTutor.

3.4. JavaSensei

Java Sensei (Zatarain Cabada et al., 2015) es un entorno de aprendizaje inteligente (ILE, por sus siglas en inglés) para la enseñanza de la programación Java a estudiantes principiantes. JavaSensei trabaja en un entorno web y aplica una estrategia de tutoría basada en afecto a través de un agente pedagógico que realiza intervenciones de tutoría de manera adaptativa considerando el estado afectivo y cognitivo del estudiante.

En la Figura 3-4 se presenta una interfaz del sistema Java Sensei donde el agente pedagógico da retroalimentación al estudiante ofreciéndole la recomendación de otros temas que podrían ser de utilidad para reforzar el aprendizaje del tema de estudio. El agente Sensei cambia sus expresiones faciales y frases dependiendo del resultado de la evaluación cognitiva y afectiva del estudiante.

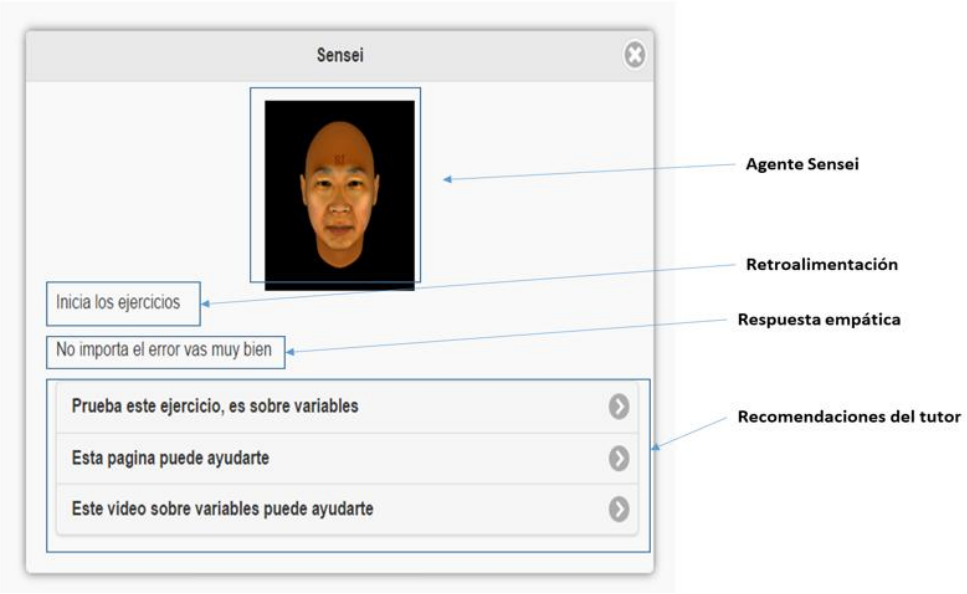


Figura 3-4. Interfaz de intervención del agente pedagógico en Java Sensei.

3.5. ITS-Debug

ITS-Debug (Carter, 2014) es un sistema que se enfoca en la enseñanza de depuración de errores de programación Java que aplica el método de enseñanza de razonamiento basado en casos. Durante la resolución de los ejercicios un estudiante visualiza un código con errores e intenta corregirlo; el sistema provee consejos cada vez que el programa del estudiante no funciona de la manera esperada. Como último recurso, el sistema proporciona un manual del tema que se está tratando y una versión correcta del programa. El estudiante durante la resolución de los ejercicios experimenta un ciclo de preguntas como: ¿He visto este error antes? ¿Qué fue lo que hice antes de arreglar el error? ¿Qué podría tener que hacer de manera diferente esta vez para corregir el error? ¿Qué tan bien funcionó esta solución en este contexto?; estas preguntas le ayudaran reflexionar sobre el error que se generó y las posibles opciones que tiene para corregirlo.

La interfaz contiene un área de codificación (área azul claro, parte superior izquierda), el estudiante mediante los botones que aparecen en la barra de menú (lado derecho) puede ejecutar el programa, restaurar el problema a su condición inicial o pasar al siguiente problema, el sistema presenta (en la parte inferior izquierda de la interfaz) el mensaje que genera el compilador Java, además también muestra (en la parte inferior derecha de la interfaz) una versión mejorada de dicho mensaje en un lenguaje más común que técnico. En la Figura 3-5 se muestra la interfaz del sistema ITS-Debug.

Figura 3-5. Interfaz de usuario de ITS-Debug.

3.6. DeBugger

DeBugger (Yoon et al., 2014) es un software en línea para múltiples jugadores (MMORPG por sus siglas en inglés) implementa la estrategia de aprendizaje colaborativo mediante juegos permitiendo a los estudiantes conversar a través del chat que ofrece la misma interfaz. *DeBugger* está integrado por un conjunto de mini juegos con el fin de lograr el aprendizaje de forma lúdica. Un ejemplo de un juego es aprender la estructura correcta de un programa Java; en este juego, el sistema presenta un programa precargado de manera desordenada para que el estudiante arrastre y suelte cada línea con el propósito de darles un ordenamiento correcto. Cuando el estudiante considera que ha terminado el ejercicio, envía su respuesta y el sistema la evalúa notificando si logró ordenar el código de forma correcta o no. El sistema no da una retroalimentación ante el error evidente del

jugador, desperdiciando un momento ideal para ofrecer conocimiento oportuno al estudiante. En la Figura 3-6 se puede observar una interfaz de este mini juego que contiene 4 secciones: arriba a la izquierda se despliega la misión (lo que debe hacer el estudiante); arriba a la derecha se muestra una ventana con información para el estudiante y los botones para enviar a evaluación el código; en la parte de abajo, se encuentran dos secciones, la izquierda contiene el conjunto de instrucciones proporcionado por el sistema y la abajo a la derecha el espacio en donde el estudiante debe arrastrar los bloques de instrucciones en orden para resolver la misión propuesta.

Figura 3-6. Interfaz de mini juego de DeBugger.

3.7. Comparación de herramientas de enseñanza de programación

En esta sección se presenta un análisis comparativo de las distintas herramientas presentadas en este capítulo.

La Tabla 3-1 resume, la inclusión o carencia de algunas características importantes consideradas en esta investigación, tales como: utilizar para la instrucción un lenguaje de programación orientado a objetos, proporcionar retroalimentación a nivel paso, usar agentes pedagógicos para la instrucción e incluir elementos de gamificación. En la última columna

se presenta un compendio de las deficiencias encontradas en los trabajos relacionados bajo el enfoque de la presente investigación.

Tabla 3-1. Comparativo de *Find Error Java* con otros trabajos relacionados.

Trabajo	Fundamento	Lenguaje	Agente	Retro-alimentación	Gamificación	Carencias
iSnap	Sugerencias basadas en datos	Algoritmos con bloques	NO	NO	NO	Enseña programación mediante bloques. Su modelo del dominio se basa en datos y no en conocimiento experto. No explica principios ni conceptos.
JALs	Aprendizaje colaborativo entre pares	Java	NO	NO	NO	Promueve el aprendizaje entre pares, no incluye enseñanza de la depuración de errores, no usa agente pedagógico, ni ofrece pistas al estudiante para resolver problemas.
JavaTutor	Reconocimiento multimodal de estados afectivos centrados en el aprendizaje	Java	NO	NO	NO	No incluye enseñanza de la depuración de errores.
Java Sensei	Basado en estados afectivos y resolución de problemas	Java	SI	SI	NO	No incluye la enseñanza de la depuración de errores.
ITS-Debug	Razonamiento basado en casos	Java	NO	NO	NO	Las pistas no son a nivel paso, los consejos pueden consistir en proporcionar un manual del tema o presentarle alguna versión de código correcto.
Debugger	Implementa técnica de aprendizaje entre pares	Java	NO	NO	SI	No se agregó conocimiento experto en la construcción de su modelo de dominio, no ofrece orientación a nivel paso.

En general, en la mayoría de los sistemas revisados, no incluyen enseñanza de la depuración de errores, ya que su enfoque principal es la enseñanza de la programación centrándose en la resolución de problemas. En cuanto a los sistemas que si consideran como tema principal la depuración de errores (ITS-Debug y Debugger), éstos no aplican en su totalidad las características planteadas en el presente trabajo de tesis tal como el uso de agentes pedagógicos, retroalimentación a nivel paso y gamificación. En el caso de Debugger, además, tampoco se utilizó conocimiento experto en la construcción de su modelo de dominio.

Capítulo 4

4. Desarrollo del proyecto

En el presente capítulo se describe el proceso completo para implementar el ambiente de aprendizaje de depuración de errores basado en ejemplos erróneos llamado *Find Error Java*.

El capítulo se divide de la siguiente manera: en la sección 4.1 se describe de manera genérica *Find Error Java*, en la sección 4.2 se presenta la metodología de desarrollo, las secciones 4.3 y 4.4 corresponden a las fases de análisis y diseño, la sección 4.5 explica la arquitectura física y la sección 4.6 exhibe el modelo de datos. La sección 4.7 corresponde a la fase de Implementación; en la sección 4.8 se despliegan las pruebas y finalmente en la sección 4.9 se muestra la fase de liberación.

4.1. Ambiente de aprendizaje Find Error Java

Find Error Java es un ambiente de aprendizaje en el que se provee al estudiante una forma sencilla y ágil de aprender a evitar o superar errores comunes que se cometen durante las primeras etapas de aprendizaje de programación Java. El sistema está diseñado para ser usado por estudiantes novatos de nivel escolar bachillerato o universidad que se encuentren estudiando cursos introductorios de programación Java.

La herramienta provee una serie de ejercicios basados en "ejemplos erróneos" en los que básicamente el estudiante debe identificar, entender y corregir el error sintáctico, semántico o lógico que contiene un pequeño programa o fragmento de código presentado.

4.2. Metodología de desarrollo

Para desarrollar el ambiente de aprendizaje *Find Error Java* fue necesario establecer los elementos clave que permitirán que el sistema funcione adecuadamente para proporcionar asistencia a los estudiantes al momento de resolver los ejercicios planteados y además poder

medir su aceptación y eficacia. Para esto, fue necesario definir cinco elementos clave para el desarrollo del sistema, estos se presentan en la Figura 4-1 y se describen a continuación:

1. Identificación de los errores comunes que cometen los estudiantes novatos al aprender a programar y creación de banco de ejercicios.
2. Diseño e implementación de estrategia de instrucción de depuración de errores mediante la adaptación a la técnica de seguimiento de ejemplos (*example-tracing*).
3. Selección e implementación de dinámicas y mecánicas correspondientes a gamificación que ayuden a motivar al estudiante a la reflexión.
4. Integración del agente pedagógico Lucy (Sosa Ochoa, 2016) a la estrategia de instrucción basada en ejemplos erróneos.
5. Diseño e implementación del plan de pruebas experimentales mediante evaluaciones de usabilidad y ganancias de aprendizaje.

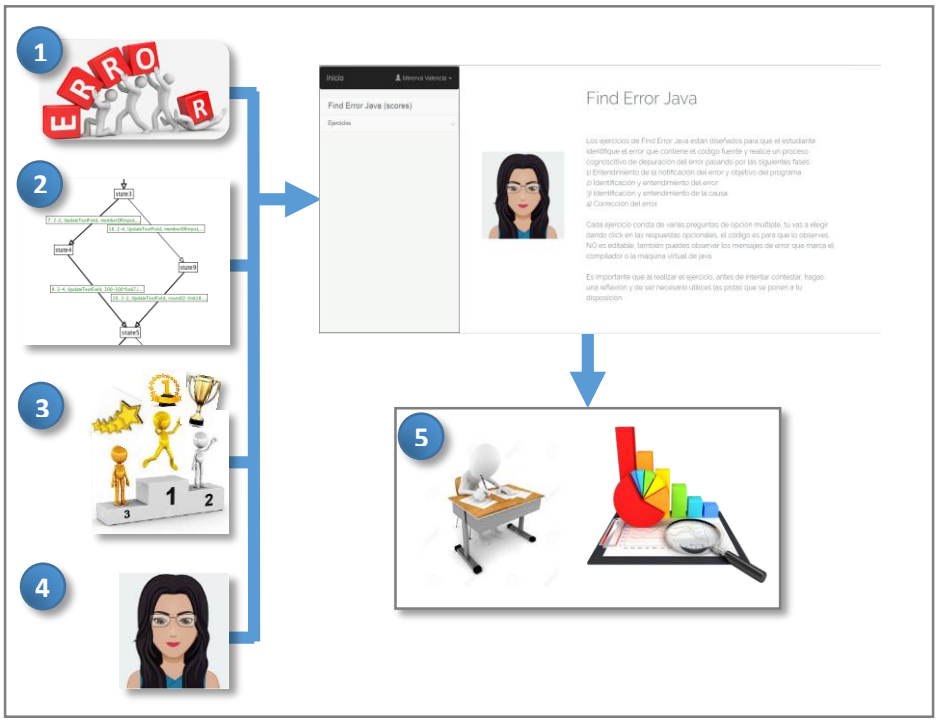


Figura 4-1. Elementos principales que fundamentan a *Find Error Java*.

Para el desarrollo del proyecto se utilizó una metodología de desarrollo iterativa e incremental cuyo flujo de ejecución se presenta en la Figura 4-2. Las secciones siguientes de

este mismo capítulo explican con detalle lo relativo a cada una de las etapas, excepto la de investigación que se detalla en los primeros 3 capítulos y la de pruebas experimentales que se detalla en el capítulo 5.

Figura 4-2. Metodología de desarrollo del proyecto.

A continuación se ofrece una breve descripción de todas las etapas que conforman la metodología.

1. **Investigación**: Incluye todas las actividades que contribuyen a la construcción de la base conceptual, metódica, procedimental y técnica del proyecto. Durante la ejecución de esta fase se realizaron investigaciones sobre sistemas tutores inteligentes, ambientes de aprendizaje, ejemplos erróneos, depuración de errores, métodos para obtener y representar el conocimiento experto y como realizar el diseño de instrucción. El resultado de esta fase se expuso en los primeros tres capítulos del presente trabajo.

2. **Análisis de requisitos**: Incluye todas las actividades que contribuyen a la definición del comportamiento del sistema. Tomando como referencia el proceso de autoría de tutores *example-tracing* así como también el estudio del pensamiento y aprendizaje del dominio mediante análisis de tareas cognitivas (CTA por sus siglas en inglés) se establecieron los requisitos funcionales del sistema, los requisitos de calidad, especificación de actores y especificación de casos de uso.
3. **Diseño de arquitectura**: En esta fase se definió la arquitectura del sistema, integrada por la especificación de los arquetipos, diagrama de contexto, vista de componentes en capas, diagrama de clases, diagrama de arquitectura física, modelo de datos y diagrama de despliegue.
4. **Implementación**: En esta fase se realizó la codificación de los componentes del sistema, la interfaz de usuario y el acceso a los datos. La codificación y creación de la base de datos relacional se realizaron con base en el análisis y diseño previamente establecido.
5. **Pruebas funcionales**: Se realizaron pruebas unitarias que aseguraron el correcto funcionamiento de cada componente del sistema así como también se realizaron pruebas funcionales que aseguraron el correcto funcionamiento del sistema completo. El resultado de esta fase de prueba dio lugar a que se realizaran correcciones y mejoras antes de proseguir con la fase siguiente.
6. **Liberación**: Se realizó la liberación de la aplicación conforme a la arquitectura diseñada, quedando listo el ambiente necesario para la realización de las pruebas experimentales.
7. **Pruebas experimentales**: Se realizó el diseño e implementación del plan de pruebas experimentales. Para medir la aceptación del usuario se utilizó el Modelo de Aceptación de la Tecnología (TAM por sus siglas en inglés) propuesto por (Davis, 1989) y para medir la ganancia de aprendizaje se utilizaron exámenes Pre-Test y Post-Test.

4.3. Análisis

En esta fase se identificaron los errores que usualmente cometen los estudiantes principiantes en cursos introductorios de programación Java del Instituto Tecnológico de Culiacán, mediante observación naturalista durante la realización de prácticas de laboratorio, revisión de portafolios de evidencia, revisión de exámenes de evaluación del curso y entrevista a estudiantes y maestros que imparten la materia.

Partiendo de los errores comunes identificados, se realizó un Análisis de Tareas Cognitivas (CTA por sus siglas en inglés) que permitió identificar los diferentes recursos cognitivos que un experto utiliza consciente o inconscientemente durante la depuración de estos errores.

A continuación se presentan los lineamientos y el proceso obtenidos a partir de este análisis:

Lineamientos

- Ofrecer al estudiante orientación sobre conceptos básicos específicos que le permitan entender el funcionamiento del lenguaje y que son necesarios para entender un error en particular así como también para poder darle solución.
- Orientar al estudiante sobre el significado de los mensajes que arroja el compilador y la máquina virtual de Java según el tipo de error encontrado.
- Para errores lógicos, ofrecer al estudiante orientación que le permita entender o interpretar el error con base en las entradas o salidas del programa.
- Durante la realización de todos los ejercicios ofrecer al estudiante orientación sobre la diferencia entre errores sintácticos, semánticos o lógicos.

Proceso de instrucción para la depuración

1. Entendimiento de la notificación del error (instrucciones del ejercicio).
2. Identificación del error en el código (línea o fragmento de código erróneo).
3. Entendimiento de la causa (conceptos, reglas o principios relacionados).
4. Corrección del error (instrucciones correctas).

El diseño de instrucción se realizó con base a los principios cognitivos descritos en la sección 2.3.1, a los errores comunes detectados, lineamientos y el proceso de instrucción para la depuración, que promueve el aprendizaje metacognitivo al propiciar que el estudiante realice un razonamiento consciente sobre cada una de las fases del proceso de depuración. Además se utilizó un enfoque constructivista en el sentido que los ejercicios consideran conocimiento previo el cual pudiera ser impreciso.

Una vez terminado el diseño de instrucción y definidos los puntos anteriormente mencionados se pudieron generar requisitos funcionales, requisitos de calidad, especificación de actores y metas, casos de uso y matriz de trazabilidad (entre requerimientos funcionales y casos de uso), los cuales se presentan a continuación.

4.3.1. Requisitos funcionales

En la Tabla 4-1, se presenta la relación de requisitos funcionales clasificados en tres niveles de prioridad: alta, media y baja. Donde la prioridad alta es para aquellos requerimientos relacionados directamente con la funcionalidad principal del sistema, prioridad media para aquellos en los que su implementación tiene un impacto indirecto en la funcionalidad principal del sistema y prioridad baja para los que no tienen un impacto en la funcionalidad principal del sistema.

Tabla 4-1. Requisitos funcionales de *Find Error Java*.

ID	Requisito	Prioridad
RF01	Administrar el perfil del estudiante.	ALTA
RF02	Autenticar el acceso a los usuarios mediante un identificador de usuario y contraseña.	ALTA
RF03	Procesar el grafo de comportamiento basado en *example-tracing* correspondiente al modelo de dominio (conocimiento experto) de cada ejercicio.	ALTA
RF04	Crear y/o actualizar en base de datos el grafo de comportamiento basado en *example-tracing* que representa el modelo del estudiante, cada vez que el estudiante realice una acción de cada ejercicio.	ALTA
RF05	Actualizar el puntaje del estudiante cada vez que el estudiante realice una acción correcta en cada ejercicio.	MEDIA
RF06	Proporcionar pistas al estudiante mediante el agente pedagógico que le ayuden a realizar acciones correctas en cada ejercicio.	ALTA
RF07	Proporcionar retroalimentación inmediata ante el éxito o fracaso del estudiante mediante un agente pedagógico.	ALTA
RF08	Intervención del agente pedagógico en caso de detectar que el alumno contesta sin reflexionar.	MEDIA
RF09	Proporcionar información respecto del uso de la herramienta misma.	BAJA
RF10	Presentar listado de las puntuaciones de todos los usuarios registrados.	ALTA

4.3.2. Requisitos de calidad

Los requisitos de calidad para *Find Error Java* corresponden a los atributos de calidad: disponibilidad, rendimiento, mantenimiento y seguridad.

En la Tabla 4-2, se presenta la relación de requisitos de calidad clasificados en dos niveles de prioridad: alta y media. Donde la prioridad alta se asigna a requerimientos que pueden afectar a la fase de pruebas del proyecto de investigación (experimentos) debido a que interfieran negativamente en la percepción del usuario en caso de no cumplirse satisfactoriamente, y prioridad media para aquellos requerimientos importantes pero no relacionados directamente con la percepción del usuario final.

Tabla 4-2. Requisitos de calidad.

ID	Atributo de calidad	Requisito	Prioridad
RC01	Disponibilidad	Tener una disponibilidad de acceso las 24/7.	ALTA
RC02	Rendimiento	El sistema debe responder a una petición del usuario en no más de 3 segundos.	ALTA
RC03	Mantenimiento	Las modificaciones a un componente no deben afectar a otros componentes.	MEDIA
RC04	Seguridad	Restringir el acceso a la información propia del usuario.	ALTA

4.3.3. Especificación de actores y metas

El sistema está enfocado en proporcionar al usuario una herramienta que le permita aprender a corregir errores de programación usando ejercicios preestablecidos, de esta forma, solamente se identificó a un usuario (estudiante) que es el actor principal que ejecutará las acciones para interactuar con el sistema. En la Tabla 4-3, se presenta al actor (estudiante) y la actividad principal que éste realiza dentro del sistema.

Tabla 4-3. Especificación de actores y metas.

Actor	Descripción	Meta
Estudiante	Persona que accede al sistema.	Usa el sistema para crear su perfil de usuario y realizar los ejercicios.

4.3.4. Casos de uso

En la Tabla 4-4 se presentan los nombres de los casos de uso clasificados en dos niveles de prioridad: alta y baja. La prioridad alta se asignó a aquellos casos de uso que se relacionan directamente con la ejecución de los ejercicios y la baja para características deseables.

Tabla 4-4. Casos de uso con prioridad.

ID	Nombre	Prioridad
CU01	Crear perfil de usuario	ALTA
CU02	Autenticación de usuario	ALTA
CU03	Realizar ejercicio de depuración de errores	ALTA
CU04	Ayuda respecto al uso del sistema mismo	BAJA
CU05	Listado de puntajes de usuarios registrados	ALTA

4.3.5. Matriz de trazabilidad entre requerimientos funcionales y casos de uso

En la Tabla 4-5 se presenta la matriz de trazabilidad de requerimientos funcionales y casos de uso, donde puede apreciarse que el CU03 tiene asignado el funcionamiento general del sistema interactuando con el usuario para cubrir 5 requisitos funcionales.

Tabla 4-5. Matriz de trazabilidad entre requerimientos funcionales y casos de uso.

ID CU	RF01	RF02	RF03	RF04	RF05	RF06	RF07	RF08	RF09	RF10
CU01	X									
CU02		X								
CU03			X	X	X	X	X	X		
CU04									X	
CU05										X

A continuación se presenta el caso de uso principal del sistema CU03, en el cual se describe el proceso de instrucción usado por *Find Error Java*. Estos ejemplos erróneos incluyen elementos de resolución de problemas ya que se le pide al estudiante que encuentre y corrija el error.

El proceso de instrucción general para realizar los ejercicios de depuración de errores consta de las siguientes fases:

- Entendimiento de la notificación del error (instrucciones).

- Identificación del error en el código.
- Entendimiento de la causa.
- Corrección del error.

ID: CU03 Realizar ejercicio de depuración de errores.	**Actor:** Estudiante.
Descripción: Practicar depuración de errores mediante estrategia basada en ejemplos erróneos.	
Precondiciones: El estudiante debe acceder a la dirección URL del sistema, autenticarse y elegir el ejercicio a realizar.	
Curso normal: 1. El sistema presenta el ejercicio mediante la información contenida en el grafo de comportamiento experto del modelo del dominio (en el punto 4.4.3 B se explica con mayor detalle), se posiciona en el primer nodo para presentar la interfaz gráfica presentada en la Figura 4-9. 2. El sistema muestra en ventana emergente las indicaciones del ejercicio. 3. El estudiante da clic en el botón cerrar indicando que leyó las instrucciones para pasar a realizar el ejercicio. 4. El sistema realiza una pregunta y ofrece pistas bajo demanda (correspondientes al nodo actual) que le pueden ayudar al estudiante a comprender la fase de instrucción y pueda contestar correctamente en el primer intento. 5. El estudiante puede solicitar pistas bajo demanda o bien puede proceder a responder la pregunta del ejercicio (opción múltiple). 6. El sistema actualiza el modelo del estudiante, almacenando sus respuestas tomando como base el grafo de comportamiento experto. 7. El agente pedagógico Lucy realiza una intervención ante una respuesta correcta, con el objetivo de reafirmar el conocimiento que se requiere transferir al estudiante. 8. El sistema avanza al siguiente nodo del grafo de comportamiento experto y se incrementa una estrella al puntaje del estudiante. En caso que el estudiante haya contestado correctamente en el primer intento se incrementa su puntaje con otra estrella extra. 9. El agente pedagógico realiza una intervención motivando al estudiante a reflexionar cuando observa que éste ha errado en 3 ocasiones anteriores. 10. El sistema pasa al siguiente nodo del grafo de comportamiento experto. En caso de ser el último nodo finaliza el ejercicio, de lo contrario, se repite el proceso desde el punto 4. Cada uno de los nodos corresponde a una fase del proceso de instrucción que utiliza *Find Error Java*.	
Alternativas: 7.a.- El agente pedagógico Lucy realiza una retroalimentación inmediata que aclara el tipo de error que cometió el estudiante y le permitirá que intente contestar nuevamente.	
Post-condición: Se habrá actualizado el puntaje (estrellas) y el modelo del estudiante (grafo de comportamiento del estudiante).	

4.4. Diseño

En esta sección se documentan los diferentes artefactos generados para la implementación del sistema *Find Error Java*.

A continuación se presentan los diferentes diagramas que se obtuvieron durante esta fase de diseño en el proceso de desarrollo de *Find Error Java*.

4.4.1. Arquetipos

En la Figura 4-3 se presentan los arquetipos (abstracciones principales) del sistema y sus relaciones.

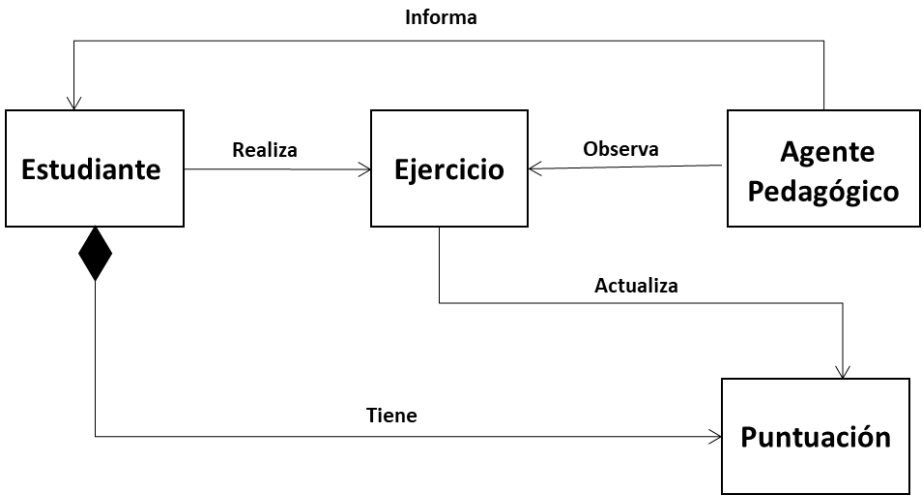


Figura 4-3. Arquetipos de *Find Error Java*.

- **Estudiante**: Es el usuario del sistema el cual debe registrarse e iniciar sesión para realizar ejercicios y/o consultar su puntuación.
- **Ejercicio**: Es el ejercicio cuyo objetivo es transferir la estrategia, conceptos y procesos de pensamiento necesarios para detectar y corregir un error en el código de un programa en Java.
- **Agente pedagógico:** Es una representación del tutor que proporciona retroalimentación al estudiante ante las acciones correctas e incorrectas de éste, le ofrece pistas y observa si su comportamiento da indicios de desinterés, para motivarlo a reflexionar y realmente intente aprender.

- **Puntuación:** Es un elemento de gamificación que tiene el objetivo de motivar al estudiante a utilizar pistas y reflexionar durante la resolución de cada ejercicio.

4.4.2. Diagrama de contexto

En la Figura 4-4 se presentan las interfaces de *Find Error Java* las cuales corresponden a la interacción del sistema con el usuario (estudiante) y con la base de datos. El estudiante usa el sistema para obtener instrucción y aprender de los temas que le presenta, mientras el sistema usa la base de datos que contiene la información del desempeño del estudiante así como lo referente al curso (ejercicios, programas, ejemplos, mensajes, etc.).

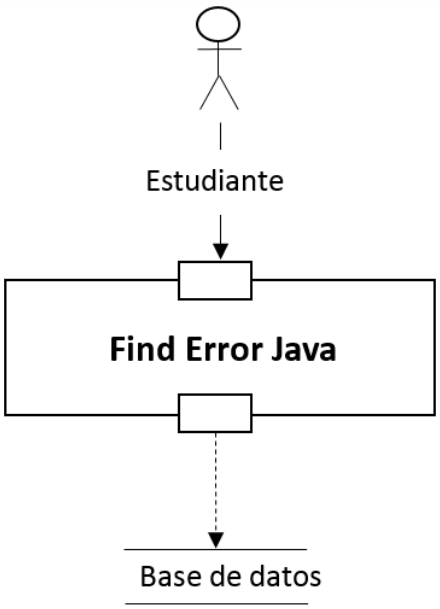


Figura 4-4. Diagrama de contexto de *Find Error Java*.

4.4.3. Arquitectura lógica

En la Figura 4-5 se muestran los componentes principales del sistema y sus relaciones. Para el diseño del sistema se eligió el estilo arquitectónico en capas que permite una organización clara e independiente de los componentes.

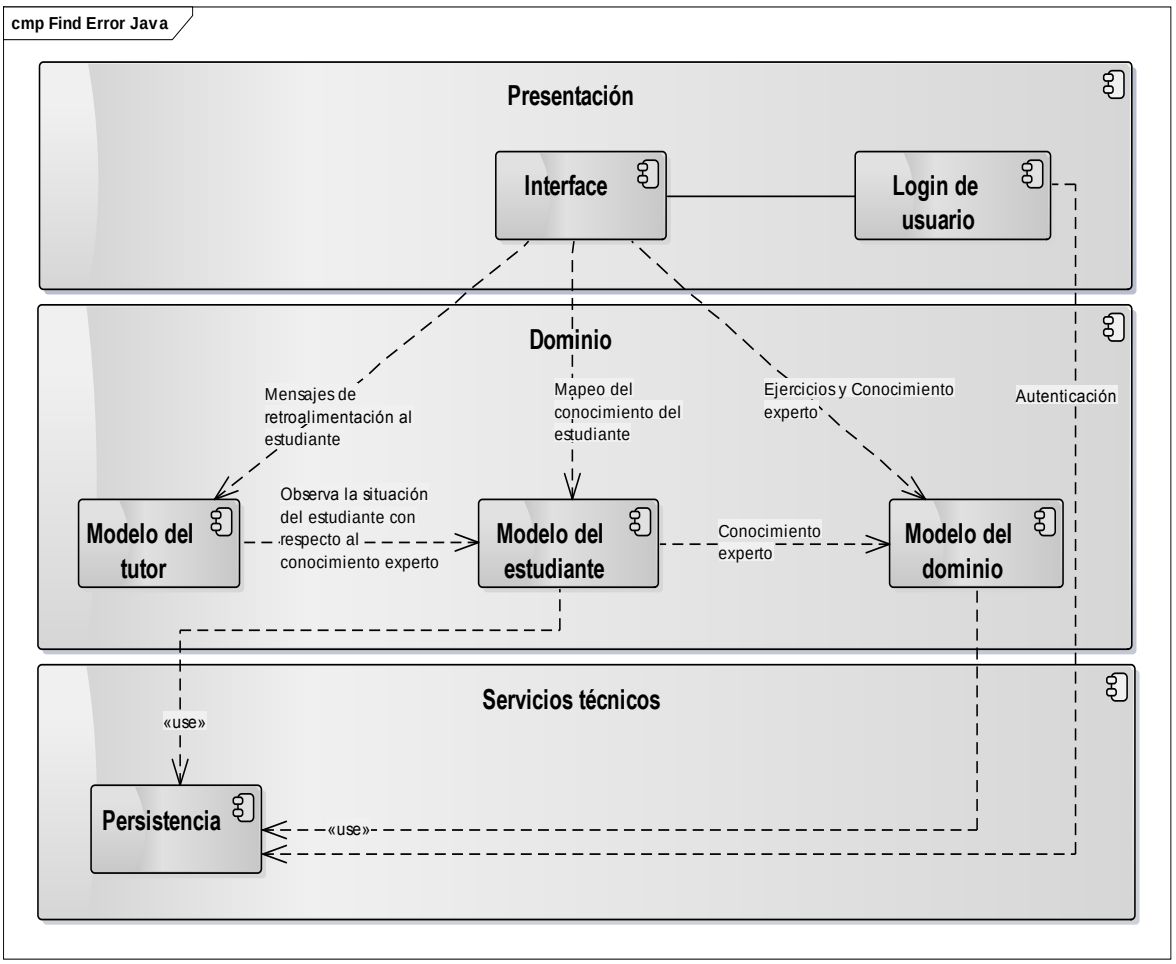


Figura 4-5. Vista de componentes en capas.

A continuación se describe cada una de las capas del sistema:

A. Capa de Presentación: Es la capa que se encarga de mostrar las interfaces con las que el usuario va a interactuar con el sistema, tales como: registro de usuario, iniciar sesión, menú principal, realización de ejercicios y consulta de puntuación. A continuación se describen las interfaces:

1) **Interfaz de bienvenida:** Esta interfaz se presenta en la Figura 4-6 y ofrece información respecto al nombre de la herramienta, objetivo pedagógico y a quien está dirigida. Mediante los enlaces que aparecen en la parte superior derecha de la interfaz de bienvenida se accede al inicio de sesión Figura 4-6 (A) o al registro de usuario Figura 4-6 (B).

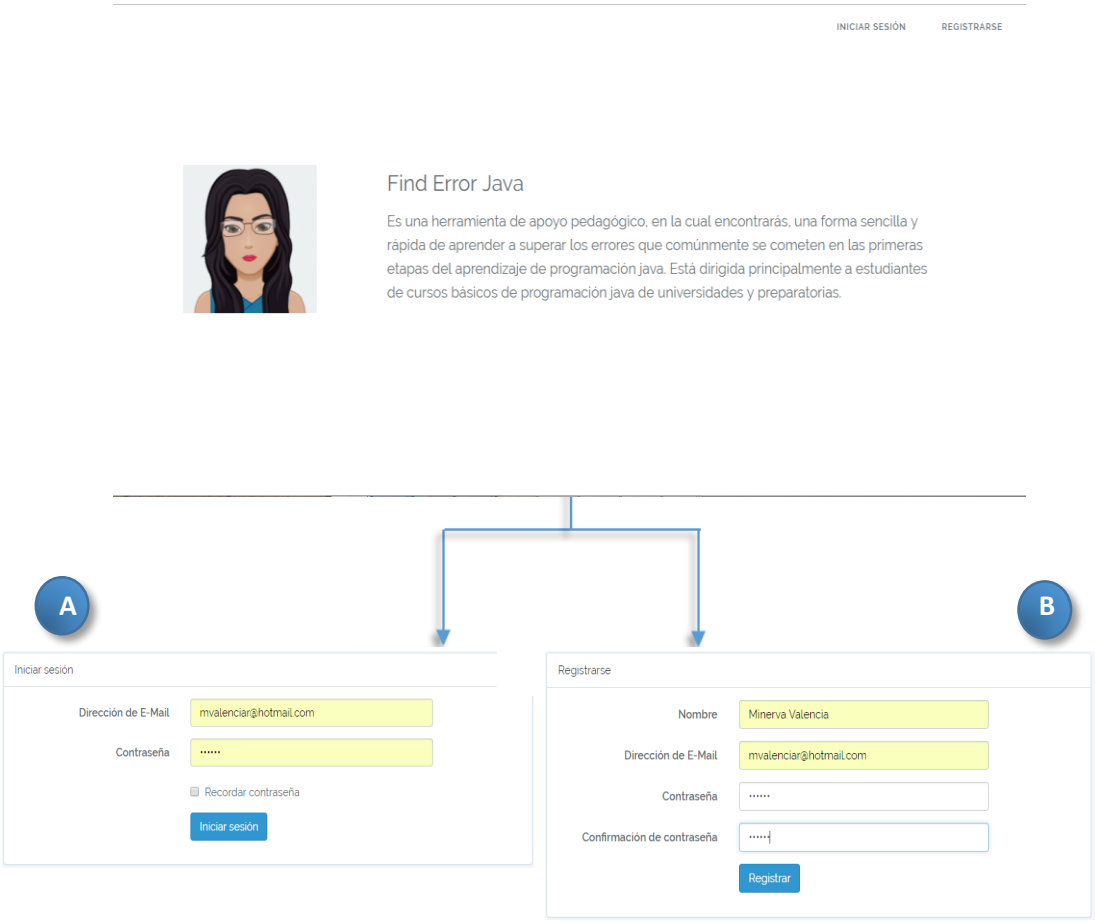


Figura 4-6. Interfaz de bienvenida.

2) **Interfaz de inicio**: En la Figura 4-7 se presenta la interfaz de inicio, donde el agente pedagógico Lucy explica brevemente el proceso de instrucción que siguen los ejercicios. Se presenta el menú principal, con las ligas a los ejercicios que el estudiante puede realizar en orden ascendente, de menor a mayor grado de complejidad.

Figura 4-7. Interfaz de inicio.

3) **Interfaz de puntuaciones:** Se utilizó un recurso de gamificación el cual consiste en ganar una estrella por cada paso del ejercicio contestado correctamente y una estrella extra si contesta correctamente en el primer intento. La filosofía aplicada fue de recompensa y no de penalización, ya que el objetivo era generar un estado de bienestar en el estudiante. Esta forma de gamificación está dirigida a motivar al estudiante a que trate de ganar la estrella extra, es decir que intente contestar una vez que esté seguro de su respuesta.

   El menú presentado en la Figura 4-7 contiene la opción "*Find Error Java* (Scores)", esta liga, lleva a la interfaz de puntuaciones presentada en la Figura 4-8, en la cual el estudiante puede comparar su puntuación con la de todos los demás participantes propiciando competencia y diversión. La información se presenta en formato de tabla especificando el lugar obtenido (consecutivo), nombre y puntuación de los estudiantes ordenándolos descendentemente, de mayor a menor puntuación alcanzada.

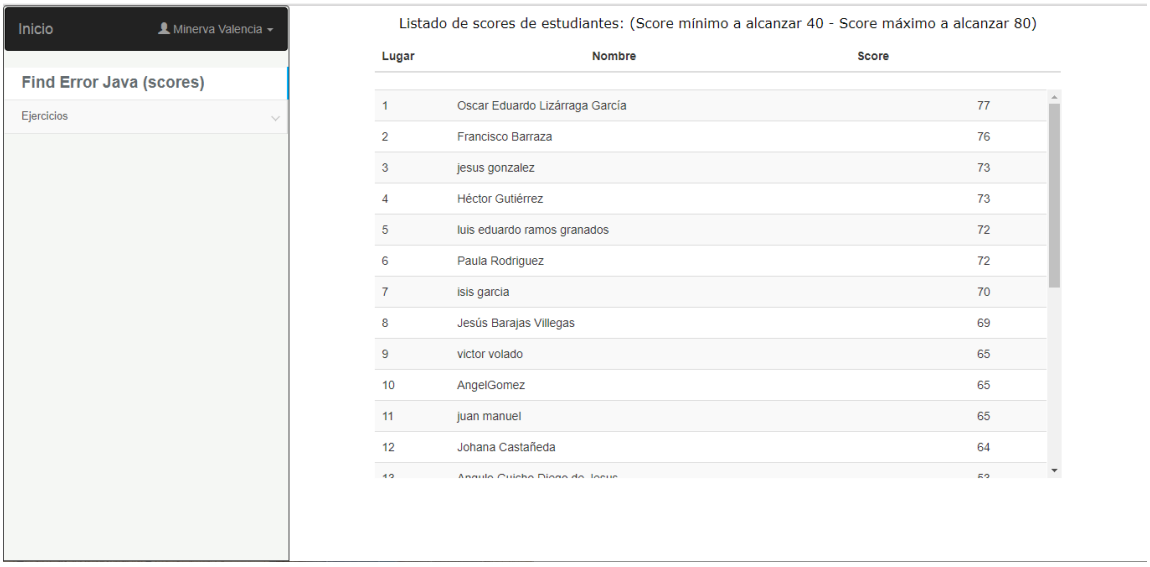


Figura 4-8. Puntuaciones de estudiantes.

4) **Interfaz de ejercicios:** Esta es la interfaz principal del sistema donde el estudiante realiza los ejercicios de depuración de errores de programación y donde se combinan diferentes técnicas de instrucción con el objetivo de darle tutoría al estudiante para que no solo aprenda a corregir el tipo de error que contiene el código del ejercicio sino también que adquiera conceptos, principios, razonamientos y habilidades que le servirán incluso para resolver otro tipo de errores. Un ejercicio puede ser resuelto en *n* pasos, donde *n* puede ser mayor o igual que 1. Un ejercicio está conformado por los siguientes atributos:
   a. **Título del ejercicio**: Es el nombre del ejercicio y hace referencia al tipo de error a tratar en el ejercicio.
   b. **Indicaciones**: Son las indicaciones generales para resolver el ejercicio. Inicia con una notificación del error "El programa no está funcionando…", seguido por una breve descripción del error, además presenta una descripción general de lo que debería estar realizando el programa si el código estuviera correcto y finalmente una breve indicación de lo que debe hacer el estudiante para resolver el error.
   c. **Código del ejercicio**: Es el código Java del ejercicio, el cual contiene el error que el estudiante debe identificar y corregir.

d. **Mensajes del compilador**: Son los mensajes de error obtenidos al compilar el código presentado en el ejercicio. En caso de tratarse de un error lógico, este atributo será ignorado en la interfaz gráfica.

e. **Mensajes de la Máquina Virtual de Java (MVJ)**: Son los mensajes de error que arroja la MVJ al ejecutar el código presentado en el ejercicio. En caso de no tratarse de un error de ejecución, este atributo será ignorado en la interfaz gráfica.

f. **Pregunta o petición de interacción**: Es una pregunta o petición dirigida al estudiante para que interactúe con la herramienta y responda eligiendo una acción optativa.

g. **Acciones optativas**: Es una lista de acciones que representan un concepto, conocimiento o razonamiento que se debe aplicar para superar el estado actual y avanzar al siguiente estado o alcanzar la finalización del ejercicio.

En la Figura 4-9 se puede observar la correspondencia entre los atributos descritos y la interfaz de ejercicios.

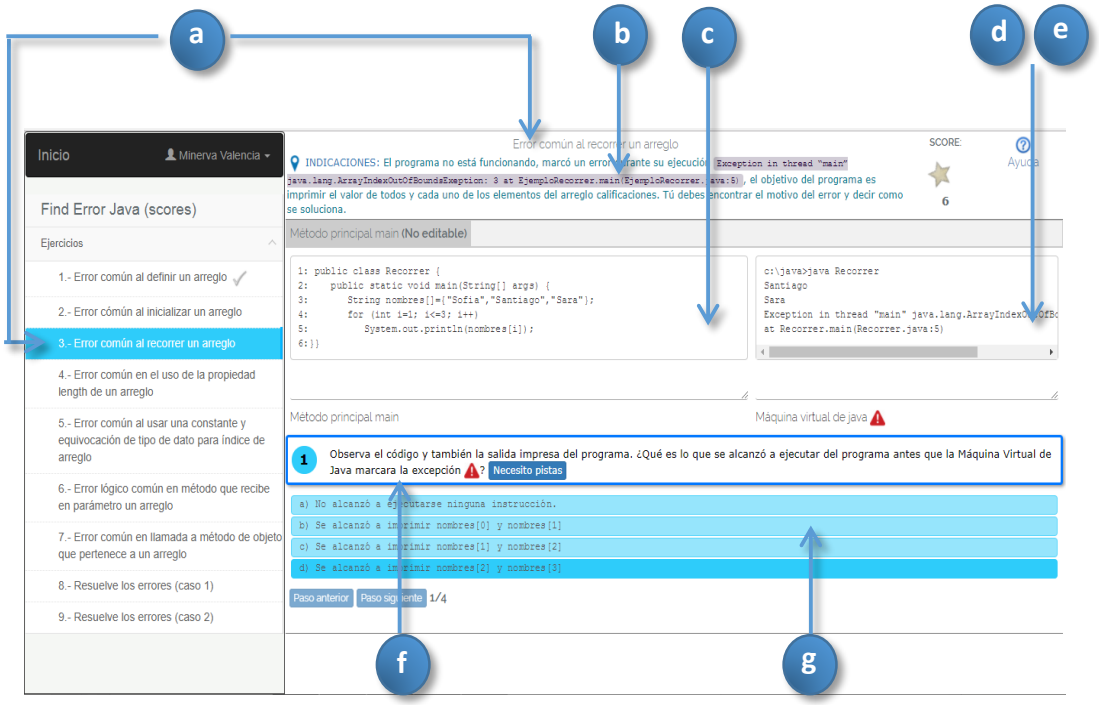


Figura 4-9. Interfaz de ejercicios.

5) **Agente pedagógico:** Se utilizó al agente pedagógico Lucy para realizar intervenciones con el estudiante durante la realización de los ejercicios. Las intervenciones del agente pedagógico pueden ser de 3 tipos como se describe a continuación:

- **Lucy proporciona pistas bajo demanda:** El estudiante puede solicitar en cada paso del ejercicio pistas, las cuales son pequeños trozos de conocimiento necesarios para contestar correctamente el paso del ejercicio, no contienen una respuesta explícita. En la Figura 4-10 se puede observar un ejemplo de una pista bajo demanda.

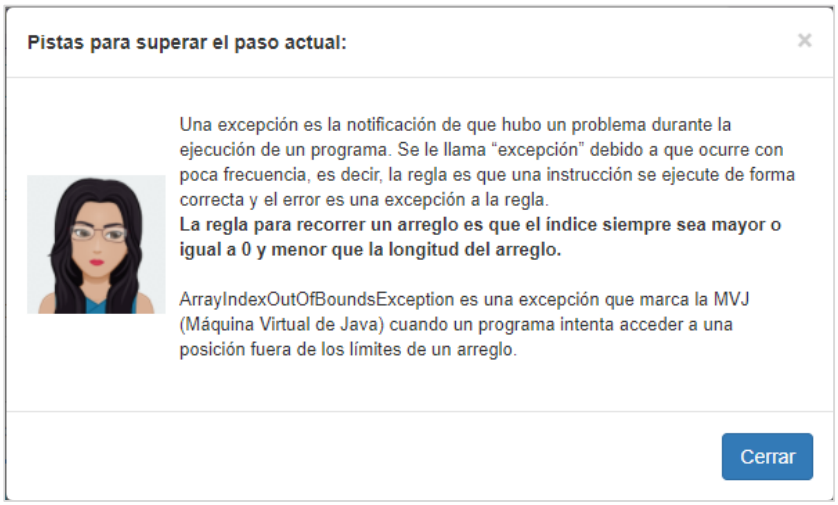


Figura 4-10. Pista bajo demanda ofrecida al estudiante.

- **Retroalimentación a nivel paso:** Lucy ofrece una retroalimentación inmediatamente después de que el estudiante comete algún error, con el fin de intentar corregir la idea errónea que haya provocado su respuesta incorrecta (ver Figura 4-11) en el momento justo cuando ocurrió.

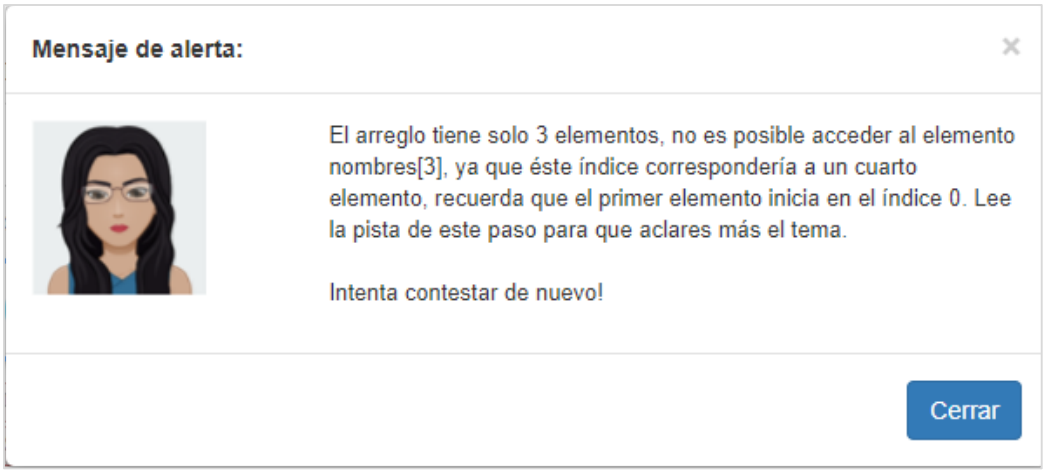


Figura 4-11. Retroalimentación inmediata respuesta incorrecta del estudiante.

También se consideró importante realizar una reafirmación del conocimiento correcto del estudiante presentándole un mensaje de éxito cuando éste contesta correctamente (Figura 4-12).

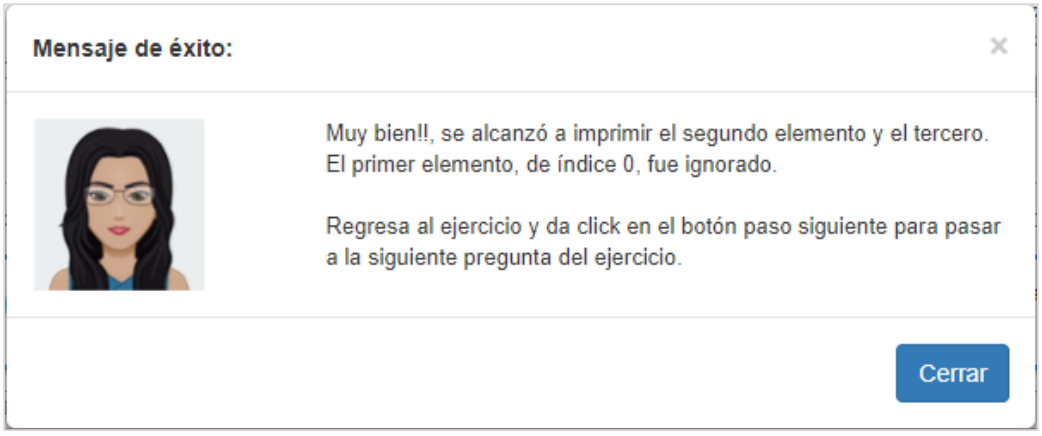


Figura 4-12. Mensaje de éxito ante respuesta correcta del estudiante.

Con el objetivo de generar empatía, Lucy cambia ligeramente de expresión facial cuando se trata de corregir al estudiante o cuando se trata de felicitarlo por una respuesta correcta. En la Figura 4-13 puede observarse la variación en la expresión facial (A) para retroalimentación debido a un error y (B) para mensaje de éxito.

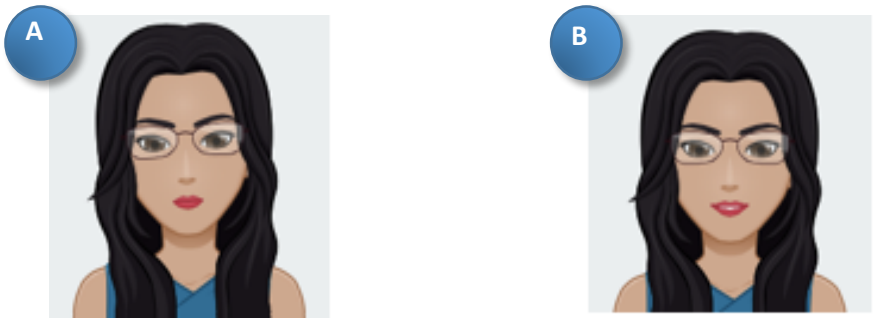


Figura 4-13. Expresiones faciales de Agente pedagógico Lucy en retroalimentación a estudiante.

- **Intervención por notificación:** Cuando el agente pedagógico considera que el estudiante no está interesado en contestar correctamente, le envía una notificación cuyo objetivo es motivarlo a leer las pistas y realizar reflexiones más profundas. No siempre envía el mismo mensaje en la notificación, en la Figura 4-14 se presenta un ejemplo.

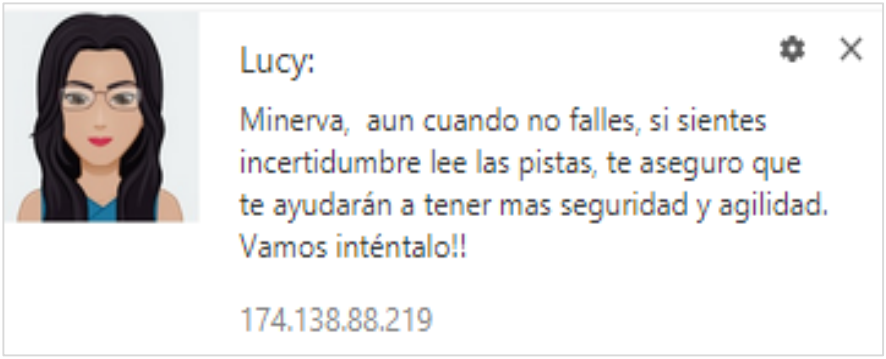


Figura 4-14. Notificación de Agente Pedagógico.

B. Dominio: Es la capa que contiene la lógica del sistema, está integrada por tres componentes principales los cuales son:

1. **Modelo del dominio**: Contiene las representaciones del conocimiento experto, respecto a los ejercicios, estrategia de depuración, conceptos y procesos de pensamiento que deben transferirse al estudiante. Este componente tiene la responsabilidad de implementar una adaptación de la técnica de seguimiento de ejemplos (*example-tracing*) para procesar un grafo

de comportamiento que incluye acciones incorrectas que pudiera cometer un estudiante durante el proceso de instrucción en depuración; así como también comportamiento óptimo, mismo que se pretende transferir al estudiante. Las acciones o comportamiento sub-óptimo de la técnica original, no fueron consideradas ya que el objetivo es mostrar un ejemplo de depuración de errores de manera óptima. El grafo de comportamiento experto del modelo del dominio contiene la estrategia de instrucción en la que se conduce al estudiante a un razonamiento consciente sobre el entendimiento de la notificación del error (instrucciones), identificación del error en el código, entendimiento de la causa y corrección del error.

En la Figura 4-15 se presenta como ejemplo los primeros 3 nodos (pasos o estados) de uno de los ejercicios de *Find Error Java*. Los nodos representan los pasos en los que se va a realizar el ejercicio y las aristas son las acciones optativas (respuestas de opción múltiple), los arcos se utilizan para representar acciones incorrectas (respuestas erróneas) en las que se requiere que el estudiante vuelva a intentar contestar, a cada arista y arco se le asocia un mensaje de retroalimentación. Una vez que el estudiante alcanza el último estado (último nodo) del grafo y conteste correctamente la pregunta, el estudiante habrá finalizado el ejercicio.

La información en el grafo de comportamiento experto, tal como, el código del ejercicio (código Java que contiene el error), mensajes del compilador, mensajes de la Máquina Virtual de Java, pregunta o petición de interacción y acciones optativas, permiten a la capa de presentación generar la interfaz de ejercicios.

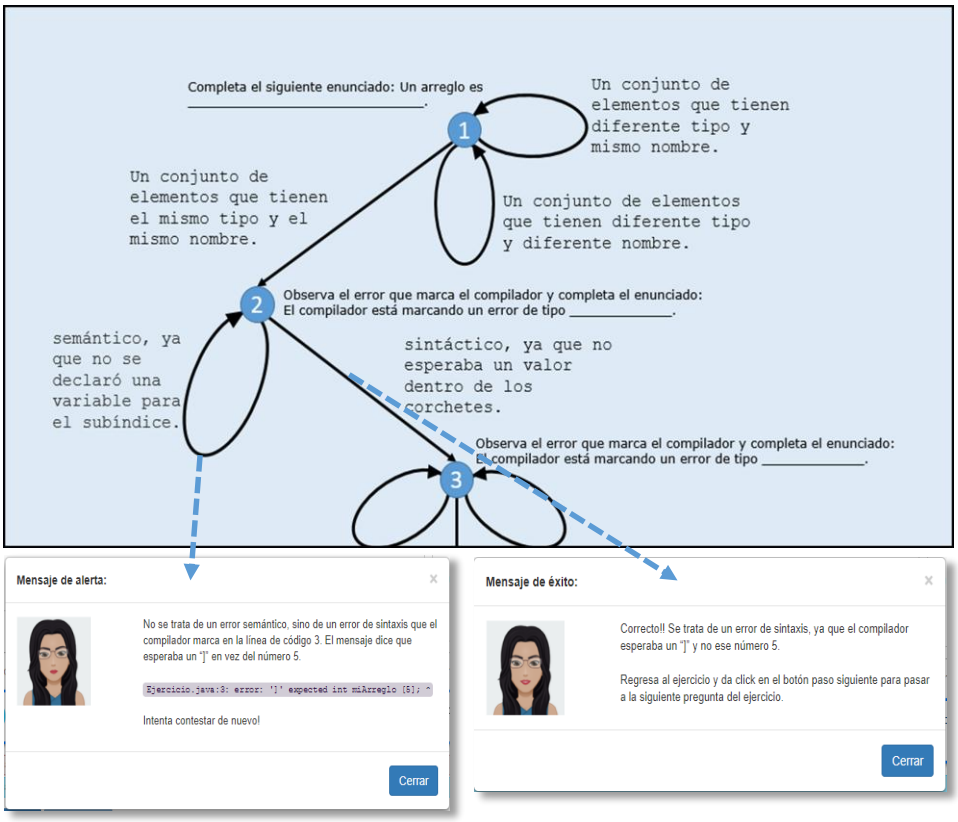


Figura 4-15. Grafo de comportamiento experto del modelo de dominio.

2. **Modelo del estudiante**: Este componente proporciona una representación del conocimiento individual del estudiante ofreciendo información respecto a los ejercicios que ha resuelto y los puntajes obtenidos. Respecto al desempeño del estudiante en cada ejercicio, toma como base el grafo de comportamiento experto para generar un grafo de comportamiento del estudiante, lo cual permite conocer, cuantos intentos realizó y qué errores cometió.

3. **Modelo del tutor**: Contiene la lógica de interacción del agente pedagógico con el estudiante. Para la intervención del agente pedagógico respecto a motivar al estudiante a reflexionar antes de contestar, se utilizó el patrón de diseño *observer* el cual consiste en definir una dependencia entre objetos de uno a muchos, de tal forma que cuando cambia de estado uno de los objetos, todas sus dependencias son notificadas y actualizadas automáticamente

(Dockins, 2017). En la Figura 4-16 se presenta el diagrama de clases correspondiente a este diseño.

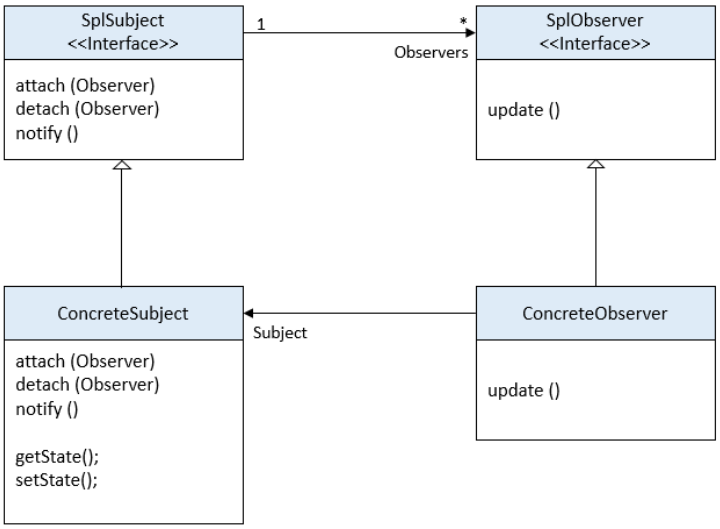


Figura 4-16. Diagrama de clases de patrón de diseño *observer*.

C. Servicios Técnicos: Es la capa que se encarga de realizar la lectura y almacenamiento de la información. La capa de servicios técnicos está conformada por modelos que el ORM de láravel (llamado *Eloquent*) permite utilizar. Los modelos tienen una correspondencia con cada una de las tablas que integran la base de datos de *Find Error Java* y son los encargados de realizar la conexión a la base de datos y ejecutar las consultas de lectura y actualización.

4.5. Arquitectura física

Find Error Java se creó con una arquitectura Cliente-Servidor, por lo que los usuarios pueden acceder desde cualquier computadora (con sistema operativo Linux, Windows o Mac OSX) con acceso a internet, mediante peticiones HTTP al servidor. En la Figura 4-17 se muestra la arquitectura física Cliente-Servidor utilizada para el sistema.

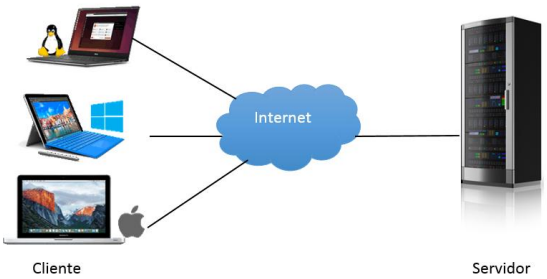


Figura 4-17. Arquitectura cliente servidor.

4.6. Modelo de datos

En la Figura 4-18 se presenta el diagrama entidad-relación correspondiente a las tablas y relaciones principales de la base de datos utilizada en *Find Error Java*. En estas tablas, entre otra información, son almacenados los grafos de comportamiento experto de cada ejercicio y los grafos de comportamiento de los estudiantes y sus puntajes.

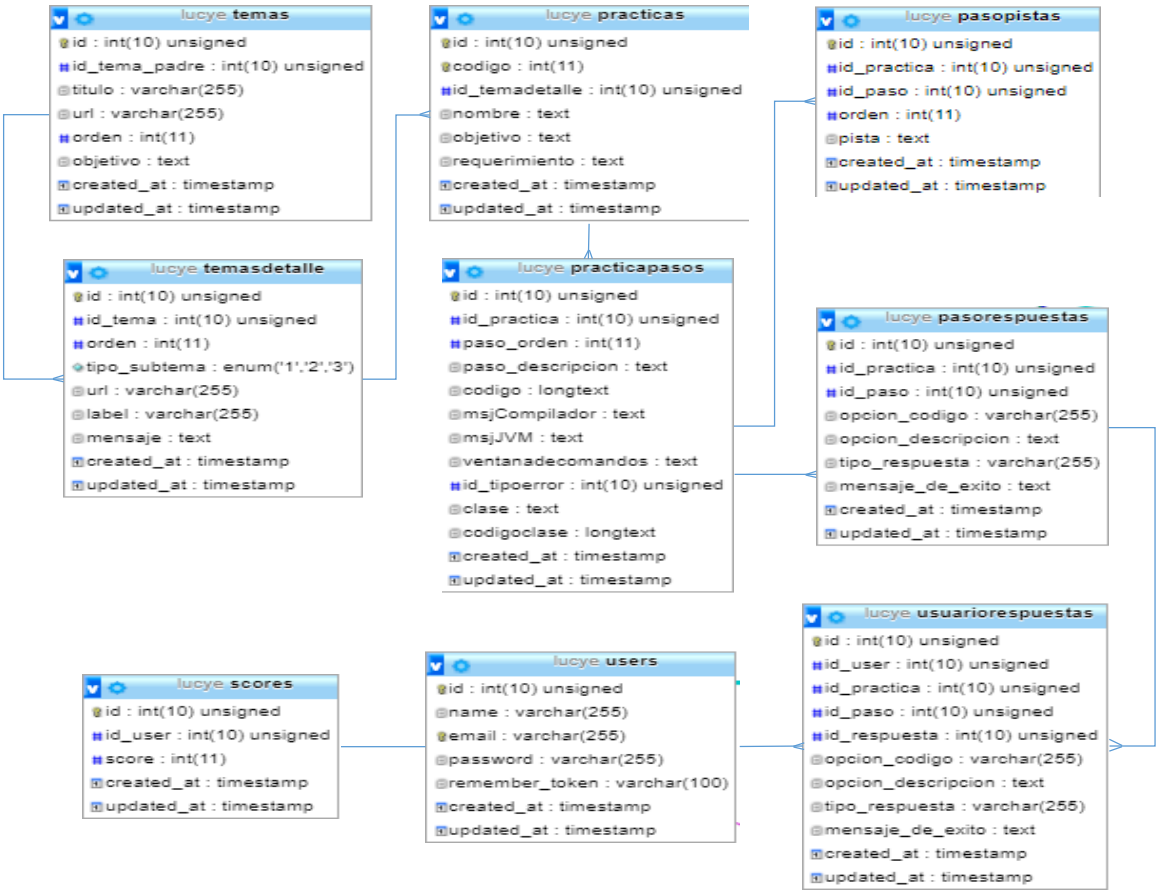


Figura 4-18. Modelo de datos.

4.7. Implementación y despliegue

Find Error Java se implementó utilizando librerías y herramientas de código libre (open source). Se eligió el framework Laravel de PHP, el cual está basado en el patrón de diseño Modelo Vista Controlador (MVC). La selección de Laravel permite, entre otras ventajas, una organización estructurada del código, promueve su reutilización y ofrece un amplio soporte a través de su documentación y basta comunidad.

Para las interfaces de usuario se utilizó HTML, Java Script, jQuery y Ajax, así como también hojas de estilo CSS y Bootstrap. Para realizar notificaciones se utilizó la librería *push.min.js*.

En cuanto a la administración de la base de datos se utilizó MySQL.

Se creó un repositorio en *Bitbucket* para el control de versiones generadas durante el desarrollo y este repositorio fue enlazado a un *droplet* (máquina virtual) de DigitalOcean donde se realizó finalmente la publicación del proyecto. Esta máquina virtual se configuró con el sistema operativo Linux de la distribución Ubuntu 16.04 y un servidor web Nginx. En la Figura 4-19 se muestra el diagrama de despliegue.

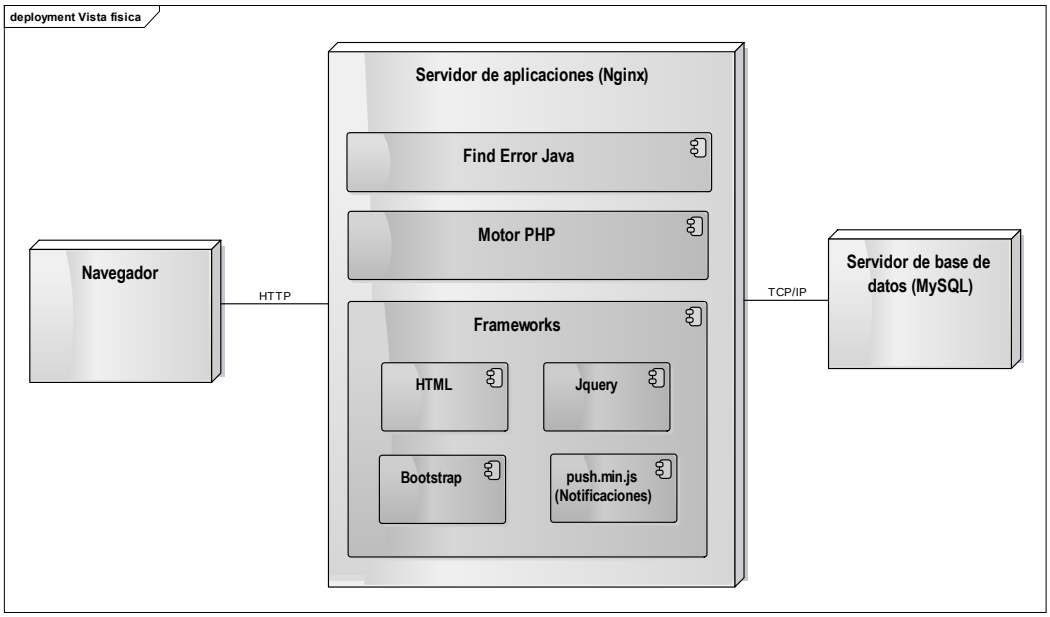


Figura 4-19. Diagrama de despliegue del sistema *Find Error Java*.

4.8. Pruebas

Las pruebas del sistema se dividieron en dos secciones: pruebas unitarias de cada uno de los componentes y pruebas de integración del sistema completo.

Los casos de pruebas unitarias diseñados se presentan en la Tabla 4-6.

Tabla 4-6. Casos de prueba.

ID	Caso de prueba	Pre condiciones	Resultado esperado
01	Perfil de usuario	No existe el usuario.	El usuario existe.
02	Autenticación de usuario.	Perfil de usuario existe y no ha iniciado sesión.	Ha iniciado sesión.
03	Realizar ejercicio.	Usuario ha iniciado sesión y ha elegido un ejercicio. El puntaje se encuentra en 0 y no hay ninguna respuesta correcta.	Respuestas correctas almacenadas, puntaje actualizado, interfaz gráfica actualizada.
04	Intervención de agente pedagógico Lucy	Usuario ha contestado erróneamente en tres ocasiones anteriores.	Muestra notificación de agente pedagógico con mensaje de motivación.
05	Mostrar ayuda respecto al uso del sistema.	Usuario ha iniciado sesión, ha elegido ejercicio y ha dado clic en botón *necesito pistas*.	Se muestra la ventana de ayuda.
06	Mostrar listado de puntajes de usuarios registrados.	Usuario ha iniciado sesión, ha elegido la opción *Find Error Java (Score).*	Se muestra la consulta de puntajes.

También se realizaron pruebas de funcionalidad para el uso concurrente de la aplicación.

A continuación se muestran las funcionalidades y flujos probados:

1. **Crear perfil**: En esta funcionalidad se registra el usuario en el sistema. Para acceder se deben registrar los datos nombre, dirección de correo electrónico, contraseña y confirmación de la contraseña. En la **Figura 4-6** (b) se presentó la interfaz.
2. **Autenticación de usuario**: Para que un usuario pueda acceder al sistema, éste debe autenticarse proporcionando su dirección de correo electrónico y contraseña. En la Figura **4-6** (a) se muestra la interfaz para iniciar sesión.
3. **Realizar ejercicio**: En la Figura 4-9 se muestra la interfaz correspondiente a un ejercicio. Cada paso del ejercicio se marca con una palomita (✓) cuando es contestado correctamente; así mismo cuando el ejercicio sea contestado

correctamente en su totalidad de pasos, en el menú, al lado derecho de la opción debe aparecer una palomita (✓) indicando que es correcto. Por cada respuesta correcta se debe incrementar una estrella al puntaje del estudiante, en caso de una respuesta correcta en el primer intento entonces se debe incrementa a la puntuación una estrella adicional.

4. **Intervención de agente pedagógico Lucy**: Se configuró al Agente Pedagógico Lucy para que al realizar una nueva pregunta de un ejercicio verifique si el estudiante ha fallado en tres ocasiones anteriores, de ser así, Lucy interpreta que el estudiante no está realmente comprometido con la comprensión del tema. La prueba consistió en comprobar que las intervenciones del agente pedagógico se realizaran de la manera configurada y que los mensajes enviados en las notificaciones cambiaran en cada ocasión. En la Figura 4-14 se puede observar un ejemplo de una notificación y en el anexo 79 se muestran los diferentes mensajes que puede enviar Lucy en cada notificación, todos éstos mensajes tienen el objetivo de motivar al estudiante a realizar reflexiones más profundas.
5. **Mostrar ayuda con relación al uso del sistema**: Se comprobó que al seleccionar el enlace *Ayuda* de la interfaz de ejercicios, apareciera una ventana emergente con información que describe brevemente el flujo y los botones que deben utilizarse para resolver un ejercicio.
6. **Mostrar tablero de puntuaciones de los usuarios**: Se comprobó que la información presentada en esta funcionalidad concuerda con los criterios definidos, los cuales son presentar una tabla con la información de los usuarios que contenga el lugar alcanzado, nombre del usuario y puntaje acumulado. Los usuarios aparecen en la tabla ordenados descendentemente de mayor a menor puntaje considerando a todos los usuarios participantes. En la Figura 4-8. ***Puntuaciones de estudiantes.*** se presenta la interfaz correspondiente a los puntajes obtenidos por los estudiantes.

4.9. Liberación

Se realizaron dos liberaciones de versión, en la primer versión se liberó la funcionalidad principal, la resolución de los ejercicios, con la finalidad de conocer el impacto de ésta en los usuarios respecto a la facilidad de uso. En la segunda versión se agregaron las recomendaciones hechas por los usuarios y las funcionalidades restantes, quedando lista la versión para la realización de los experimentos.

Capítulo 5

5. Pruebas

En este capítulo se muestran los resultados del estudio realizado para medir dos aspectos de *Find Error Java*: aceptación de usuario y ganancia de aprendizaje significativo.

El capítulo se divide en cuatro secciones, en la sección 5.1 se describe la población, en la sección 5.2 se presenta la planificación de las pruebas, la sección 5.3 corresponde a la evaluación de la aceptación de *Find Error Java* por parte del usuario y finalmente en la sección 5.4 se muestra la evaluación de *Find Error Java* para lograr una ganancia de aprendizaje significativo.

5.1. Población

El estudio se realizó en el Instituto Tecnológico de Culiacán, y participaron 46 estudiantes (9 mujeres y 37 varones) en un rango de 18-21 años, pertenecientes a las carreras Ingeniería en Sistemas Computacionales e Ingeniería en Tecnologías de la Información y Comunicación.

5.2. Planificación del estudio

Para la ejecución del estudio se diseñaron una serie de actividades que se muestran en la Tabla 5-1.

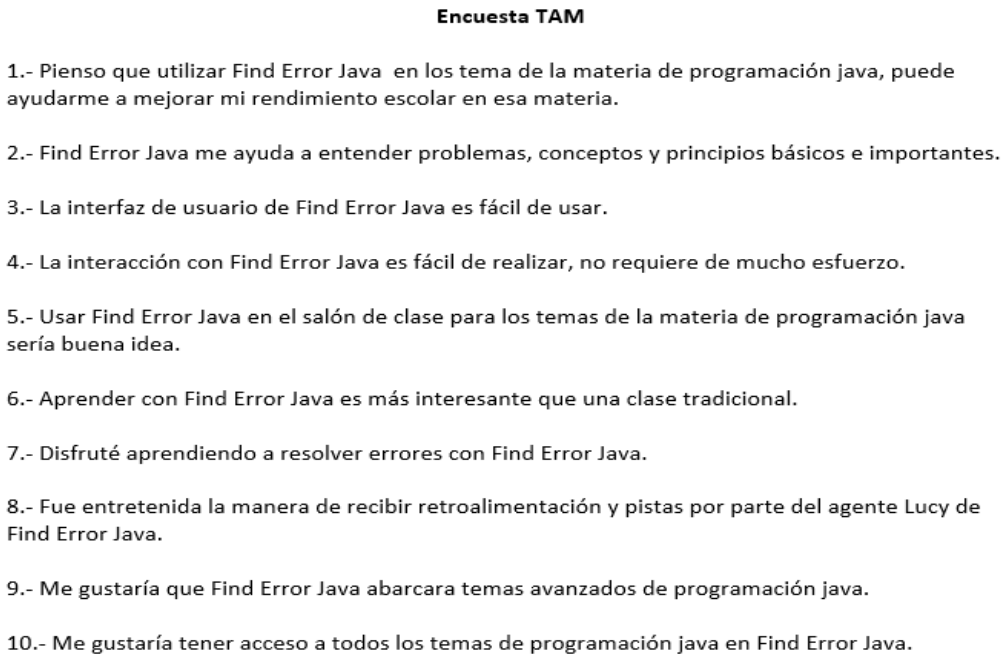
Tabla 5-1. Listado de actividades para la realización del estudio.

Orden	Actividad
1	Selección de la muestra (estudiantes que participan en el estudio).
2	Distribución de estudiantes en la muestra en dos grupos: experimental y control.
3	Aplicación de examen Pre-Test a ambos grupos.
4	Utilización de *Find Error Java* en grupo experimental.
5	Aplicación de encuesta TAM a grupo experimental.
6	Aplicación de examen Post-Test a ambos grupos.

5.3. Evaluación de aceptación por el usuario

Para evaluar la aceptación de usuario de *Find Error Java* se utilizó el Modelo de Aceptación de la Tecnología (TAM por sus siglas en inglés) propuesto por (Davis, 1989) el cual propone que la utilidad percibida y la facilidad de uso percibida por el usuario son determinantes en la aceptación del software o herramienta tecnológica analizada. El modelo propone la evaluación de las variables: utilidad percibida, facilidad de uso percibida, diversión o disfrute, actitud de uso e intención de uso.

Se diseñó un cuestionario (ver Figura 5-1), el cual evalúa las variables que propone el modelo TAM.



Encuesta TAM

1.- Pienso que utilizar Find Error Java en los tema de la materia de programación java, puede ayudarme a mejorar mi rendimiento escolar en esa materia.

2.- Find Error Java me ayuda a entender problemas, conceptos y principios básicos e importantes.

3.- La interfaz de usuario de Find Error Java es fácil de usar.

4.- La interacción con Find Error Java es fácil de realizar, no requiere de mucho esfuerzo.

5.- Usar Find Error Java en el salón de clase para los temas de la materia de programación java sería buena idea.

6.- Aprender con Find Error Java es más interesante que una clase tradicional.

7.- Disfruté aprendiendo a resolver errores con Find Error Java.

8.- Fue entretenida la manera de recibir retroalimentación y pistas por parte del agente Lucy de Find Error Java.

9.- Me gustaría que Find Error Java abarcara temas avanzados de programación java.

10.- Me gustaría tener acceso a todos los temas de programación java en Find Error Java.

Figura 5-1. Preguntas del cuestionario TAM.

Cada una de las preguntas del cuestionario fue contestada mediante un valor ponderado correspondiente a una escala Likert de 1 a 7, cuyos valores y significados se presentan en la Figura 5-2.

Figura 5-2. Escala de Likert para evaluación de aceptación de *Find Error Java*.

5.3.1. Análisis detallado de la evaluación TAM

En la Figura 5-3 se presenta, por cada estudiante, el valor promediado de las respuestas de la encuesta TAM. Es decir, se calculó por cada estudiante el promedio considerando los valores Likert de las respuestas de cada una de las 10 preguntas de la encuesta. Esta gráfica permite observar a simple vista que la mayoría de los estudiantes que usaron *Find Error Java* tuvieron una apreciación parecida. Además se puede observar que la mayoría de las respuestas de los estudiantes sugieren una buena aceptación ya que el 82.6% de los estudiantes reportó un promedio por encima de 5 de la escala de Likert utilizada.

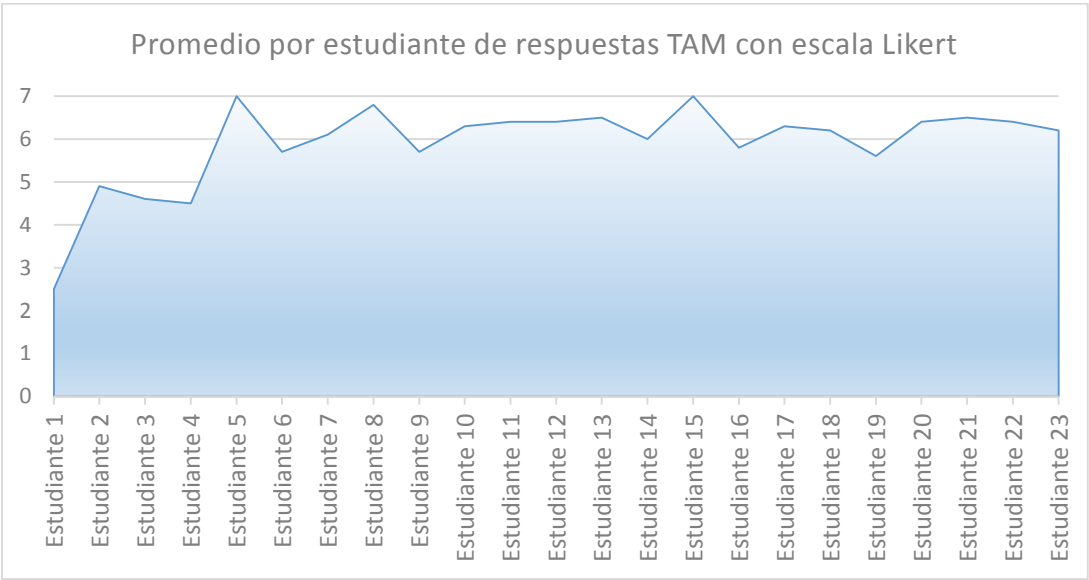


Figura 5-3. Promedio por estudiante de la evaluación TAM.

En la Figura 5-4 se presenta el promedio obtenido por cada pregunta considerando las respuestas de los 23 estudiantes, en esta gráfica, se puede observar que la mayoría de las respuestas de los estudiantes para cada una de las preguntas sugieren una buena aceptación, ya que el promedio estuvo por encima del valor 5 de la escala de Likert utilizada.

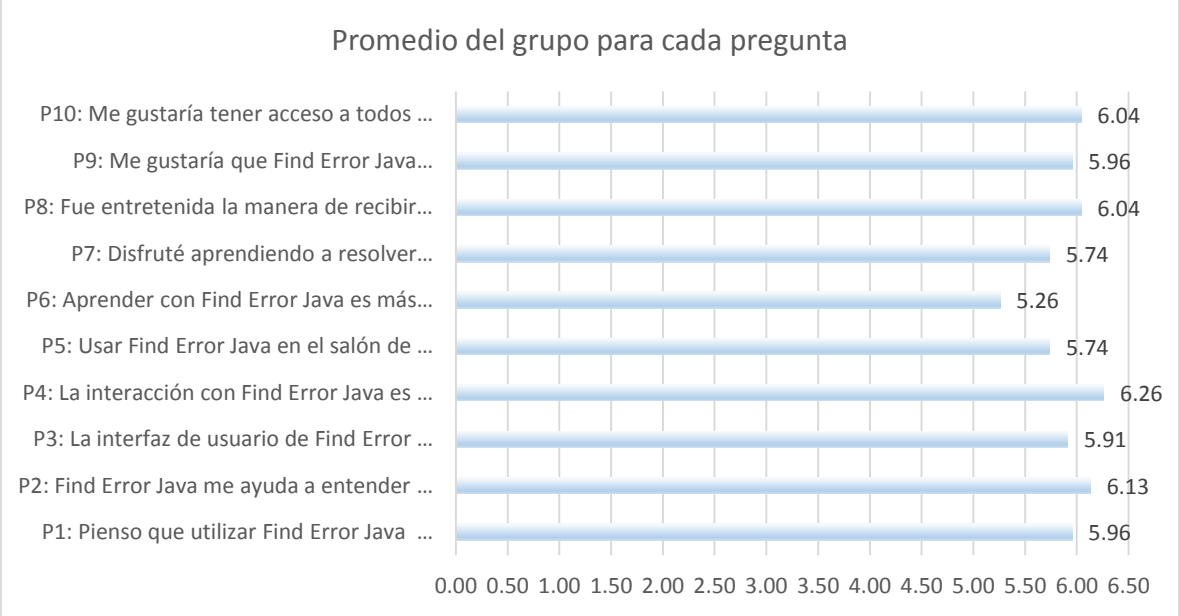


Figura 5-4. Promedio de respuestas de encuesta TAM por pregunta.

5.3.2. Análisis e interpretación del coeficiente de Alfa de Cronbach

Los datos obtenidos por la encuesta TAM se transcribieron a una hoja de cálculo (Excel) , en donde se realizaron los cálculos correspondientes a la fórmula estadística del coeficiente Alfa de Cronbach, misma que se presenta en la fórmula 1. En esta fórmula, K es el número de ítems, S_i^2 es la sumatoria de varianzas de los ítems, S_T^2 es la varianza de la suma de los ítems y α es el coeficiente de Alfa de Cronbach.

$$\alpha = \frac{K}{K-1}\left[1 - \frac{\sum S_i^2}{S_T^2}\right] \qquad (1)$$

Para la interpretación del coeficiente se aplican los criterios que se muestran en la Tabla 5-2.

Tabla 5-2. Valores del coeficiente Alfa de Cronbach.

Coeficiente alfa	Interpretación
α > 0.9	Excelente
α > 0.8	Bueno
α > 0.7	Aceptable
α > 0.6	Cuestionable
α > 0.5	Pobre
α < 0.5	Inaceptable

El coeficiente Alfa de Cronbach que se obtuvo para *Find Error Java* fue de 0.93 clasificado como un resultado excelente.

5.4. Efectividad en ganancias de aprendizaje significativo

Para la evaluación de la efectividad en ganancia de aprendizaje, se decidió abordar con *Find Error Java* el tema de arreglos, ya que este tópico es generalmente un problema para los estudiantes durante los cursos básicos de programación, persistiendo estos problemas incluso en cursos posteriores a los básicos.

Para medir y comparar la ganancia de aprendizaje entre alumnos que usan *Find Error Java* y los que no lo usan, la muestra de 46 estudiantes se dividió en dos grupos, el primero denominado grupo experimental y el segundo denominado grupo de control.

A los 46 estudiantes, previo a la utilización de *Find Error Java*, se les aplicó una primera evaluación denominada Pre-Test la cual consta de dos secciones, la primera relacionada a la depuración de errores y la segunda relacionada a la escritura de código correcto. Posteriormente, se realizó una sesión de 45 minutos con el grupo experimental para que resolvieran los ejercicios de depuración de errores de *Find Error Java*. En la Figura 5-5 puede observarse a los alumnos del grupo experimental utilizando *Find Error Java*.

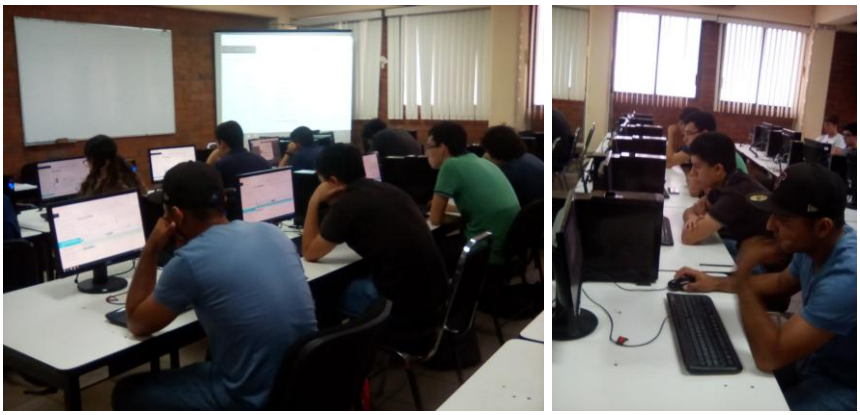


Figura 5-5. Estudiantes del grupo experimental usando *Find Error Java*.

Posteriormente se aplicó a ambos grupos una segunda evaluación denominada Post-Test.

5.4.1. Ganancia de aprendizaje en relación a depuración de errores

A continuación se presentan los resultados obtenidos para la primera sección de las evaluaciones Pre-Test y Post-Test relacionadas con la depuración de errores.

En la Figura 5-6 se presenta un comparativo de las calificaciones obtenidas en el Pre-Test y Post-Test correspondientes al grupo experimental, el cual utilizó *Find Error Java*. La mayoría de los estudiantes (78.2%) del grupo experimental, presentaron un incremento en su calificación en el examen posterior al uso de la herramienta.

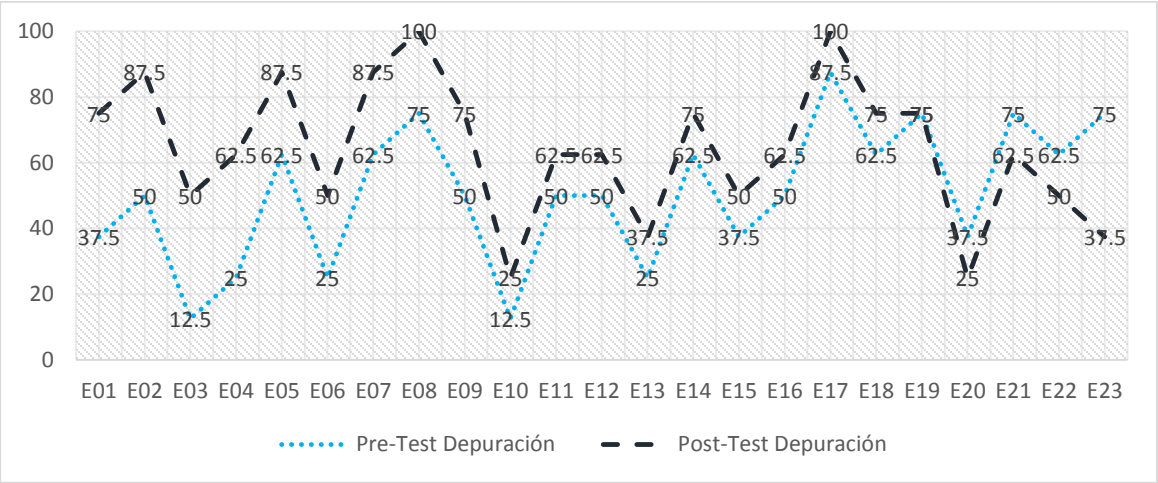


Figura 5-6. Comparativo grupo experimental en relación a depuración.

En la Figura 5-7 se presenta un comparativo de las calificaciones obtenidas en el Pre-Test y Post-Test correspondientes al grupo de control, el cual no utilizó *Find Error Java*. En este caso fue el 26% del grupo de control, el que presentó un incremento en su calificación.

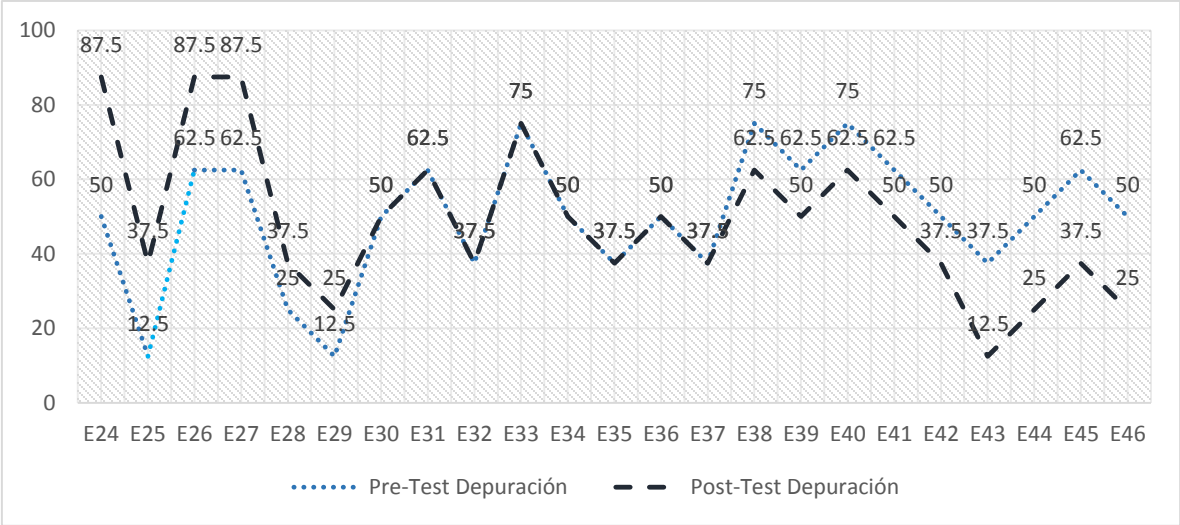


Figura 5-7. Comparativo grupo de control en relación a depuración.

5.4.2. Ganancia de aprendizaje en relación a escritura de código correcto

A continuación se presentan los resultados obtenidos para la segunda sección de las evaluaciones Pre-Test y Post-Test relacionadas con la escritura de código correcto.

En la Figura 5-8 se presenta un comparativo de las calificaciones obtenidas en el Pre-Test y Post-Test correspondientes al grupo experimental, el cual utilizó *Find Error Java*. El 47.8% del grupo experimental, presentó un incremento en su calificación en el examen posterior al uso de la herramienta.

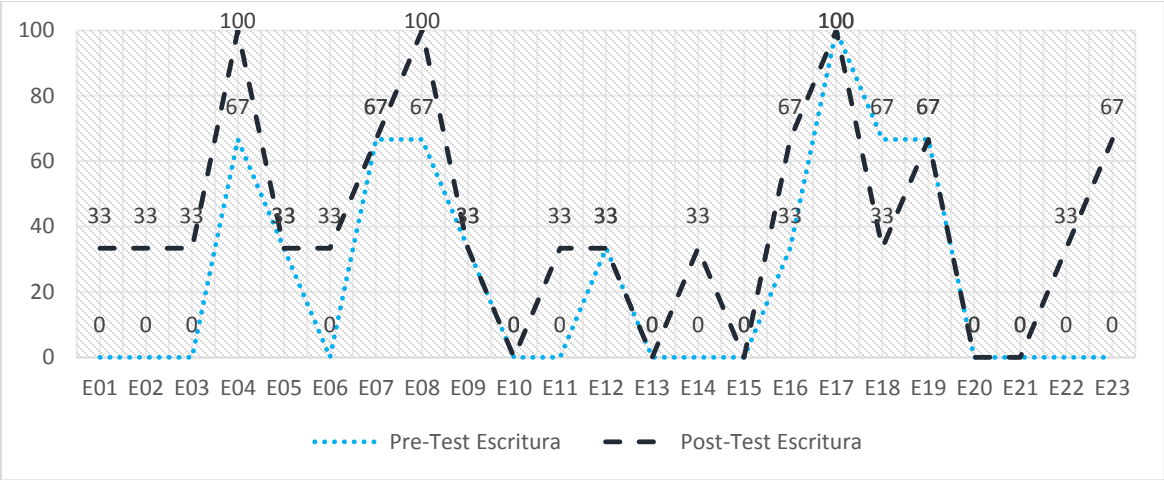


Figura 5-8. Comparativo grupo experimental en relación a escritura de código correcto.

En la Figura 5-9 se presenta un comparativo de las calificaciones obtenidas en el Pre-Test y Post-Test correspondientes al grupo de control, el cual no utilizó *Find Error Java*. En este caso fue el 30.4% el que presentó un incremento en su calificación.

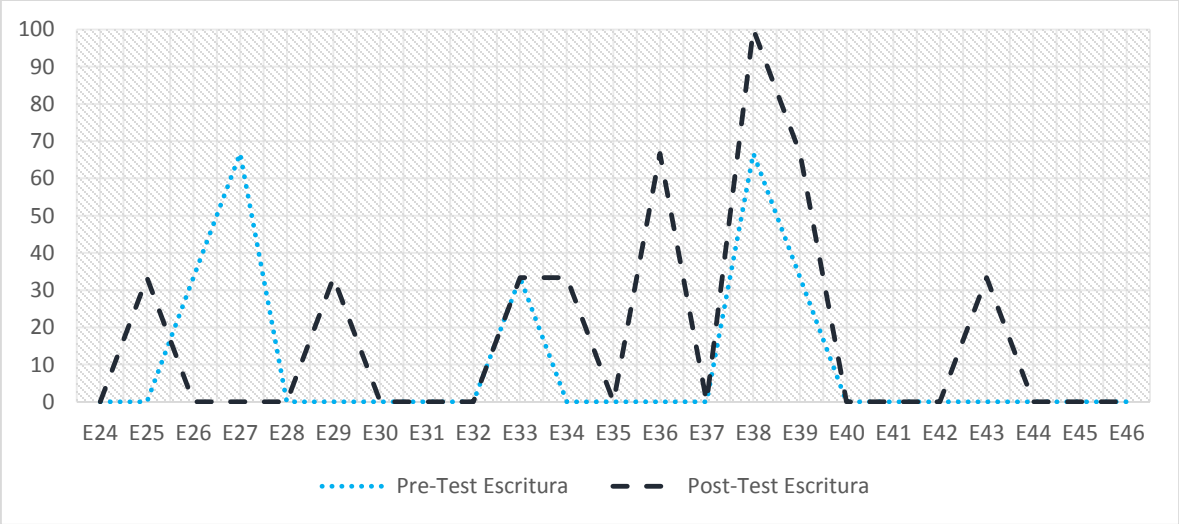


Figura 5-9 Comparativo grupo de control en relación a escritura de código correcto.

Aunque en ambas secciones, el grupo experimental presentó un porcentaje mayor de estudiantes que mejoraron sus calificaciones, la sección relacionada con depuración de errores fue la que destacó en su efecto positivo de reforzamiento. Con la intención de confirmarlo, se calcularon los porcentajes correspondientes a la cantidad de estudiantes que presentaron variaciones positivas, nulas y negativas en el tema relacionado con depuración de errores, estas variaciones representan respectivamente, que mejoraron, se mantuvieron o empeoraron.

Las Figura 5-10 y Figura 5-11 muestran las proporciones en que los estudiantes presentaron dichas variaciones. Puede observarse claramente que el grupo experimental tiene una variación positiva mucho mayor que el grupo de control, lo que confirma que *Find Error Java* tuvo un efecto positivo en los estudiantes.

Figura 5-10. Variaciones en grupo experimental.

Figura 5-11. Variaciones en grupo de control.

Capítulo 6

6. Conclusiones y trabajo futuro

La actividad de depuración de errores ha sido reconocida como una actividad importante en el proceso de programación. Y se ha reconocido el fuerte impacto que ésta tiene en los programadores novatos.

Los errores de programación pueden utilizarse naturalmente como ejemplos erróneos, los cuales a su vez ofrecen una gran oportunidad de aprendizaje cuando son bien utilizados en el desarrollo de temas y actividades de aprendizaje. El uso de ejemplos erróneos genera conflicto cognitivo y siembran curiosidad en el estudiante (animan a la reflexión e investigación).

Find Error Java fue diseñado tomando como base los errores comunes de estudiantes de las carreras de Ingeniería en Sistemas Computacionales e Ingeniería en Tecnologías de la Información y Comunicación del Instituto Tecnológico de Culiacán, sin embargo al comparar estos errores con los errores comunes descritos en la literatura se encontraron grandes similitudes. Por lo que su aportación no solo aplica a esta población en particular.

La instrucción que se diseñó para *Find Error Java* considera los diferentes recursos cognitivos que un experto utiliza consciente o inconscientemente durante la depuración de errores abarcando todos los recursos que cada caso necesita y no centrándose en un solo aspecto del problema, como por ejemplo centrando la tutoría solo en el entendimiento de los mensajes de error o solo en el aprendizaje de conceptos teóricos.

El objetivo de la gamificación fue motivar al estudiante para que intentara hacer reflexiones profundas antes de contestar cada paso de los ejercicios, además se requería propiciar un ambiente de competencia con la finalidad de divertirlos, no estresarlos.

Para lograr motivarlos y no estresarlos, se utilizó la obtención de estrellas, una estrella por cada paso de ejercicio contestado correctamente y una estrella extra si se contestó correctamente en el primer intento. No se incluyeron penalizaciones, además debido a la naturaleza de los ejercicios (respuestas de opción múltiple) y a la cantidad de pasos que

integran a los ejercicios, cualquier estudiante que realizara todos los ejercicios, su puntuación mínima sería de 40 puntos. El estudiante realmente compite por puntos extras (otros 40 puntos posibles de ganar si responden bien en el primer intento).

Durante la realización de los ejercicios con *Find Error Java*, los estudiantes mostraron interés en saber en qué lugar estaban logrando posicionarse (accediendo de vez en cuando a la interfaz de puntuaciones). Lo cual nos sugiere que se estaba logrando el objetivo, motivarlos a ganar puntos extras, que solo se lograrían si comprenden y reflexionan antes de contestar.

Los resultados obtenidos en los experimentos donde se midió la usabilidad mediante una encuesta TAM y efectividad en ganancias de aprendizaje mediante evaluaciones de conocimiento Pre-Test y Post-Test demuestran que el uso de una herramienta como *Find Error Java*, puede ser de gran utilidad para que los estudiantes de programación aprendan a depurar los errores del código que se generan cuando se diseña un programa para resolver un problema.

6.1. Aportaciones

A continuación se presentan las aportaciones del presente proyecto:

- Diseño de estrategia de instrucción para la enseñanza de depuración de errores sustentada en lineamientos y proceso de depuración obtenidos mediante ACT.
- Adaptación del grafo de comportamiento de para la tutoría de ejemplos erróneos.
- Implementación de ambiente de aprendizaje para la depuración de errores que integra a la estrategia de instrucción un agente pedagógico y elementos de gamificación.
- Banco de ejercicios y ambiente de aprendizaje.

6.2. Trabajo futuro

Como trabajo futuro a continuación se presentan puntos que se consideran de importancia para ser incluidos en la herramienta:

- Agregar ejercicios de mayor dificultad.
- Agregar funcionalidad que permita darle seguimiento paso a paso al código erróneo, para los casos de errores donde el seguimiento del flujo de control del programa y la revisión del valor de variables sea importante.
- Agregar un área de evaluación donde el estudiante realice ejercicios de depuración de errores (estudiados previamente) mediante codificación libre, de tal manera que se fortalezca la precisión del modelo del estudiante y se le de tutoría de los temas que aún no haya terminado de comprender.
- Agregar responsabilidad al módulo tutor para que adapte la tutoría al estudiante con mayor precisión.

Bibliografía

Ahmadzadeh, M., Elliman, D., & Higgins, C. (2005). An analysis of patterns of debugging among novice computer science students. *ACM SIGCSE Bulletin*, *37*(3), 84. https://doi.org/10.1145/1151954.1067472

Aleven, V., Mclaren, B. M., Sewall, J., & Koedinger, K. R. (2009). A new paradigm for intelligent tutoring systems : Example-tracing tutors. *International Journal of Artificial Intelligence in Education*, *19*, 105–154. Retrieved from http://iospress.metapress.com/index/X3760U67H22LM111.pdf

Aleven, V., Mclaren, B. M., Sewall, J., Velsen, M. Van, Popescu, O., & Demi, S. (2016). Example-Tracing Tutors : Intelligent Tutor Development for Non-programmers. https://doi.org/10.1007/s40593-015-0088-2

Baylor, A. L. (2000). Beyond Butlers: Intelligent Agents as Mentors. *Journal of Educational Computing Research*, *22*(4), 373–382. https://doi.org/10.2190/1EBD-G126-TFCY-A3K6

Baylor, A. L., & Kim, Y. (2005). Simulating instructional roles through pedagogical agents. *International Journal of Artificial Intelligence in Education*, *15*(1), 95–115. https://doi.org/10.1007/BF02504991

Benander, A. C., & Benander, B. A. (1989). An Analysis of Debugging Techniques. *Journal of Research on Computing in Education*, *21*(4), 447–455. https://doi.org/10.1080/08886504.1989.10781893

Borasi, R. (1994). Capitalizing on Errors as "Springboards for Inquiry": A Teaching Experiment. *Journal for Research in Mathematics Education*, *25*(2), 166–208. https://doi.org/10.2307/749507

Boulay, B. du, O'Shea, T., & Monk, J. (1981). The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-Machine Studies*, *14*(3), 237–249. https://doi.org/10.1016/S0020-7373(81)80056-9

Brusilovsky, P., Pesin, L., & Zyryanov, M. (1993). Towards an adaptive hypermedia component for an intelligent learning environment. *Human-Computer Interaction*. Retrieved from http://www.springerlink.com/index/P2HPW70PU226WK3L.pdf%5Cnpapers://3a07567f-5013-4eec-86cd-a5ef539fd065/Paper/p4780

Carter, E. E. (2014). An Intelligent Debugging Tutor For Novice Computer Science Students. *PhD Thesis*.

Chan, T.-W., & Chou, C.-Y. (1997). Exploring the Design of Computer Supports for Reciprocal Tutoring. *International Journal of Artificial Intelligence in Education*, *8*, 1–29.

Chorney, A. I. (2012). Taking The Game Out Of Gamification. *Dalhousie Journal of Interdisciplinary Management*, *8*(1). https://doi.org/10.5931/djim.v8i1.242

Clark, R. E., Feldon, D. F., van Merriënboer, J. J. G., Yates, K. a, & Early, S. (2008). Cognitive Task Analysis. *Handbook of Research on Educational Communications and Technology (3rd Ed.)*, 577–593. Retrieved from http://www.cogtech.usc.edu/publications/cta_chapter_2008.pdf

Confrey, J. (1990). What Constructivism Implies for Teaching Author. *Journal for*

Research in Mathematics Education, *Volumen 4*, 107–122.
Cross, J. H., Hendrix, T. D., & Barowski, L. A. (2011). Combining dynamic program viewing and testing in early computing courses. *Proceedings - International Computer Software and Applications Conference*, 184–192. https://doi.org/10.1109/COMPSAC.2011.31
Davis, F. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, *13*(3), 319–340. https://doi.org/10.1016/S0305-0483(98)00028-0
Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). From game design elements to gamefulness: Defining gamification. *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments - MindTrek '11*, 9–11. https://doi.org/10.1145/2181037.2181040
Dillenbourg, P., & Self, J. A. (1992). People power: A human-computer collaborative learning system. *Intelligent Tutoring Systems*, *608*(i), 651–660. https://doi.org/10.1007/3-540-55606-0_75
Dockins, K. (2017). *Design Patterns in PHP and Laravel. Apress.*
Du Boulay, B. (1986). Some Difficulties of Learning to Program. *Journal of Educational Computing Research*, *2*(1), 57–73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9
Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, *18*(2), 93–116. https://doi.org/10.1080/08993400802114508
Gee, J. P. (2003). *What video games have to teach us*. *Journal of Chemical Information and Modeling* (Vol. 53). https://doi.org/10.1017/CBO9781107415324.004
Ginat, D., & Shmalo, R. (2013). Constructive use of errors in teaching CS1. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education. Monograph*, *4*, 353. https://doi.org/10.1145/2445196.2445300
Graesser, A. C., Person, N. K., & Harter, D. (2001). Teaching Tactics and Dialog in AutoTutor. *International Journal of Artificial Intelligence in Education*, *12*(3), 257–279. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Teaching+Tactics+and+Dialog+in+AutoTutor#0
Gugerty, L., & Olson, G. (1986). Debugging by skilled and novice programmers. *ACM SIGCHI Bulletin*, *17*(4), 171–174. https://doi.org/10.1145/22339.22367
Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. *ACM SIGCSE Bulletin*, *35*(1), 153–156. https://doi.org/10.1145/792548.611956
Jackson, J., Cobb, M., & Carver, C. (2005). Identifying Top Java Errors for Novice Programmers. *Proceedings Frontiers in Education 35th Annual Conference*, T4C–24–T4C–27. https://doi.org/10.1109/FIE.2005.1611967
Jenkins, T. (2002). On the difficulty of learning to program. *In Proceedings for the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, Loughborough*, 27–29.
Kopp, S., Gesellensetter, L., Krämer, N. C., & Wachsmuth, I. (2005). A conversational agent as museum guide - Design and evaluation of a real-world application. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3661 LNAI*, 329–343.

https://doi.org/10.1007/11550617_28
Kopp, V., Stark, R., & Fischer, M. R. (2008). Fostering diagnostic knowledge through computer-supported, case-based worked examples: Effects of erroneous examples and feedback. *Medical Education*, *42*(8), 823–829. https://doi.org/10.1111/j.1365-2923.2008.03122.x
Lai, C. H., Lin, W. C., Jong, B. S., & Hsia, Y. T. (2013). Java assist learning system for assisted learning on facebook. *Proceedings - 2013 Learning and Teaching in Computing and Engineering, LaTiCE 2013*, 77–82. https://doi.org/10.1109/LaTiCE.2013.10
Lawrance, J., Bogart, C., Burnett, M., Bellamy, R., Rector, K., & Flemming, S. D. (2013). How programmers debug, revisited: An information foraging theory perspective. *IEEE Transactions on Software Engineering*, *39*(2), 197–215.
Malone, T. W., & Lepper, M. R. (1987). Making learning fun: A taxonomy of intrinsic motivations for learning. *Aptitude Learning and Instruction*. https://doi.org/10.1016/S0037-6337(09)70509-1
McCall, D., & Kölling, M. (2015). Meaningful categorisation of novice programmer errors. *Proceedings - Frontiers in Education Conference, FIE*, *2015–Febru*(February). https://doi.org/10.1109/FIE.2014.7044420
Melis, E. (2004). Erroneous Examples as a Source of Learning in Mathematics. *IADIS International Conference: Cognitive and Exploratory Learning in the Digital Age*, 311–318.
Melis, E., Sander, A., & Tsovaltzi, D. (2010). How to Support Meta-Cognitive Skills for Finding and Correcting Errors? *2010 AAAI Fall Symposium Series*, *3*, 64–68. Retrieved from http://www.aaai.org/ocs/index.php/FSS/FSS10/paper/viewFile/2179/2590
Mitrovic, A., & Ohlsson, S. (1999). Evaluation of a constraint-based tutor for a database language. *International Journal of Artificial Intelligence in Education*, *10*, 238–256. Retrieved from http://www.uic.edu/depts/psch/ohlson-1.html%5Cnhttp://ir.canterbury.ac.nz/handle/10092/327
Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: the good, the bad, and the quirky – a qualitative analysis of novices' strategies. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08)*, *40*, 163. https://doi.org/10.1145/1352135.1352191
Price, T. W. (2017). iSnap : Towards Intelligent Tutoring in Novice Programming Environments, 483–488.
Rath, A., & Brown, D. E. (1995). Conceptions of human-computer interaction: a model for understanding student errors. *Journal of Educational Computing Research*, *12*(4), 395–409. https://doi.org/10.2190/30P8-5RUB-F6KR-11G1
Raymer, R. (2011). Gamification?: Using Game Mechanics to Enhance eLearning. *eLearn Magazine*.
Sleeman, D., Putnam, R. T., Baxter, J., & Kuspa, L. (1986). Pascal and high school students: A study of errors. *Journal of Educational Computing Research*, *2*(1), 5–23. https://doi.org/10.2190/2XPP-LTYH-98NQ-BU77
Sosa Ochoa, C. (2016). *Agente Pedagógico con estrategia "Aprendiz Cognoscitivo" para un ambiente de programación colaborativo*. Instituto Tecnológico de Culiacán.
Sottilare, R. A., Graesser, A. C., Hu, X., & Goldberg, B. S. (2014). *Design Recommendations for Intelligent Tutoring Systems* (Vol. 2).

Stott, A., & Neustaedter, C. (2013). Analysis of gamification in education. *Surrey*, 1–8. Retrieved from http://carmster.com/clab/uploads/Main/Stott-Gamification.pdf

Vanlehn, K. (2006). The Behavior of Tutoring Systems. *Int. J. Artif. Intell. Ed.*, *16*(3), 227–265. Retrieved from http://dl.acm.org/citation.cfm?id=1435351.1435353

Vinner, S. (1983). Concept definition, concept image and the notion of function. *International Journal of Mathematical Education in Science and Technology*, *14:3*, 293–305.

Wiggins, J. B., Boyer, K. E., Baikadi, A., Ezen-Can, A., Grafsgaard, J. F., Ha, E. Y., … Wiebe, E. N. (2015). JavaTutor: An Intelligent Tutoring System that Adapts to Cognitive and Affective States during Computer Programming. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, 599–599. https://doi.org/10.1145/2676723.2691877

Woolf, B. P. (2010). *Building Intelligent Interactive Tutors Student-centered strategies for revolutionizing e-learning*. *Morgan Kaufmann*. https://doi.org/10.1007/BF02680460

Yoon, I., Kang, E., & Kwon, O. (2014). DeBugger Game: Mobile Virtual Lab for Introductory Computer Programming Courses. *Proceedings of the 2014 American Society for Engineering Education Zone IV Conference*.

Zatarain Cabada, R., Barron Estrada, M. L., Gonzalez Hernandez, F., & Oramas Bustillos, R. (2015). An Affective Learning Environment for Java. *2015 IEEE 15th International Conference on Advanced Learning Technologies*, 350–354. https://doi.org/10.1109/ICALT.2015.53

Zhang, D., Zhao, J. L., Zhou, L., & Nunamaker, J. F. (2004). Can e-learning replace classroom learning? *Communications of the ACM*, *47*(5), 75–79. https://doi.org/10.1145/986213.986216

Anexos

A. Mensajes de notificación

ID	Mensaje
1	<nombre del usuario>, "es muy importante que NO contestes al azar, date la oportunidad de practicar con estos ejercicios reflexionando e intentándolo, tu puedes, lo lograrás!!."
2	<nombre del usuario>, "aun cuando no falles, si sientes incertidumbre lee las pistas, te aseguro que te ayudarán a tener más seguridad y agilidad. Vamos inténtalo!!"
3	<nombre del usuario>, "ánimo!!, te recomiendo reflexiones primero en el objetivo del programa, después enfócate encontrar la instrucción errónea, luego procede a entender la causa para poder proseguir con la corrección del código."
4	<nombre del usuario>, "en caso que no tengas certeza, puedes leer las pistas, te darán seguridad para responder, ánimo!!"
5	<nombre del usuario>, "has fallado en varias intentos, tómate tu tiempo para leer detenidamente las pistas y reflexionar cada paso, puedes regresarte a pasos anteriores o a los primeros ejercicios, sigue practicando !!".