Universidad de las Ciencias Informáticas

Facultad 3

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Título: Sistema para el cálculo de relaciones de recurrencia

Autor(es):

Yoel Pimienta Rivas

Manuel Arias Furones

Tutor:

MSc. Yalice Gámez Batista

Junio, 2016

La Habana

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

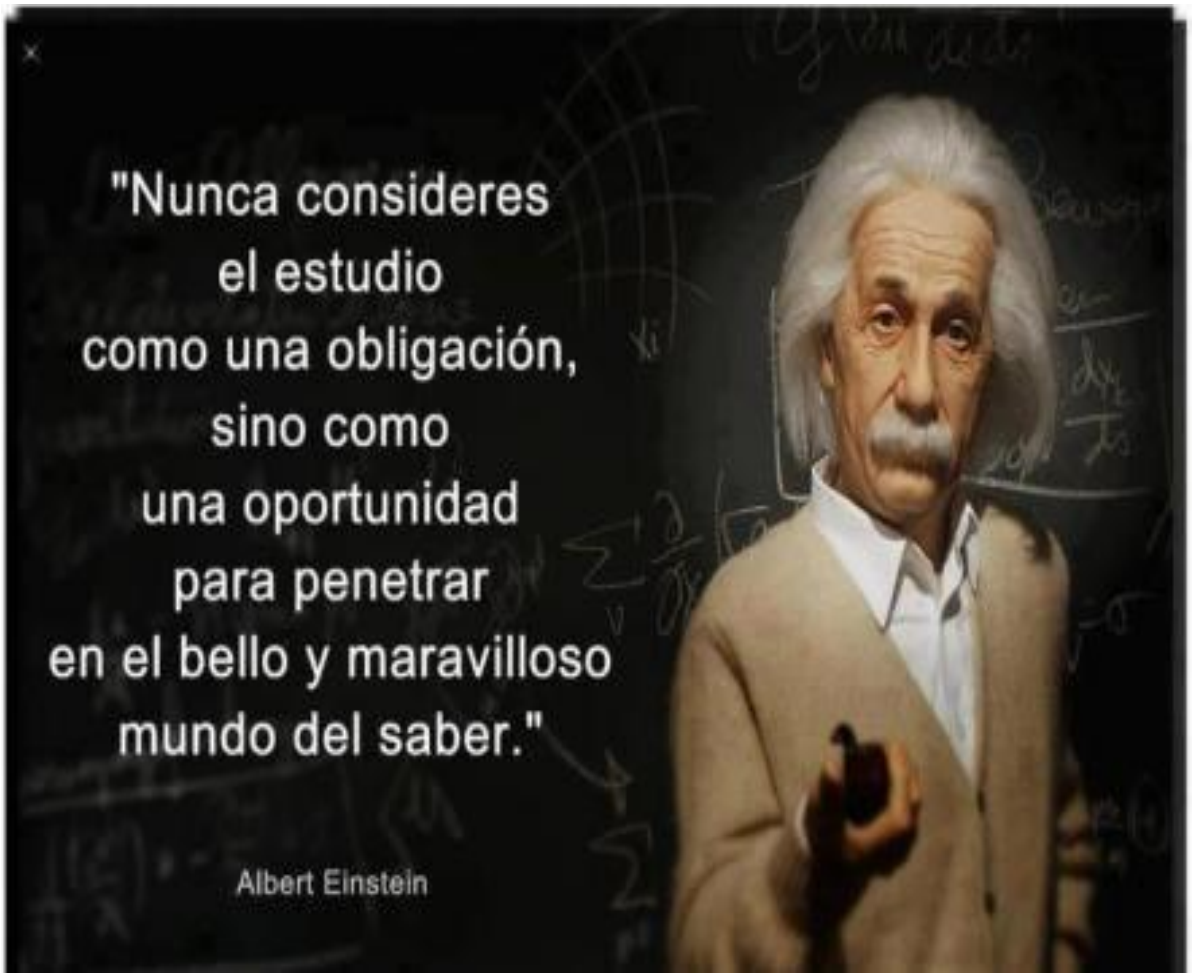
Para que así conste firmamos la presente a los _____

_____ _____

Autor: Yoel Pimienta Rivas Autor: Manuel Arias Furones

_____

Tutora: MSc. Yalice Gámez Batista


"Nunca consideres
el estudio
como una obligación,
sino como
una oportunidad
para penetrar
en el bello y maravilloso
mundo del saber."
Albert Einstein

DEDICATORIA

Manuel:

Dedico esta tesis a mi mamá Luisa Furones Acosta y a mi papá Manuel Arias Perdomo, por ser las personas más maravillosas que tengo en esta vida, por confiar en mí y apoyarme durante estos cinco años de universidad.

A mi hermana Milena Arias Furones y Thalía Vargas.

A mis tías Santa, Ana Iris, Elis, Daysis, Tatica y Ener que, aunque estén lejos las amo.

A mis tíos Evidio y Luis.

A mis abuelas Milagros y Evidia.

A Yanet, Henry, Yusimí, Oldenis, Yairon y Dailét que más que amigos son hermanos para mí, por todo el apoyo brindado y por compartir momentos inolvidables.

A Hairam Vargas y Yanet Águila, que son como mi familia.

A todos mis compañeros de aula, por apoyarme y pasar momentos maravillosos.

A Carlos, Neysis, Asprón, Pedro, Joenis y todo el piquete de la gran familia.

A todas las personas que de una u otra manera contribuyeron en mi carrera; a todos los quiero mucho, nunca los olvidaré.

Yoel:

Dedico esta tesis a mis padres, mi mamá Neosladia Rivas Rodríguez y a mi papá Yoel Pimienta Álvarez por todo lo que me han enseñado en la vida, por el apoyo incondicional que me brindan y por estar ahí cada vez que los necesito.

A mi hermanita linda Rosalia Pimienta Rivas por ser mi fiel compañera para toda la vida.

A mi abuelita Miriam por todo el cariño y el amor que siempre me ha dado desde pequeño.

A mi abuela Magdalena.

A mi novia preciosa Gabriela por este maravilloso tiempo junto.

A mi abuelo Arnulfo, por ser una de las personas que más quiero en esta vida, por ser mi fiel amigo, por todo lo que me ha ayudado en mis años de carrera, sin él no estuviera aquí ahora.

A mi amigo Carlos por estar siempre a mi lado en estos cinco años

A mi mejor amiga Neysi, a Pedrito, a Joenis, y a todos mis compañeros de aula por estos años juntos.

A todas esas personas que estuvieron a mi lado y me apoyaron en mi carrera.

AGRADECIMIENTOS

Manuel:

Agradecer a mi tutora Yalice Gámez Batista por todo su apoyo y por confiar en mí, y darme la maravillosa oportunidad de convertirme en ingeniero.

A mi compañero de tesis Yoel Pimienta Rivas, que es como un hermano para mí, por estar siempre presente en las buenas y en las malas, apoyarme y darme buenos consejos.

A mis padres y mi hermana por el apoyo incondicional en todo momento.

A Yairon, Dailét, Yanet y La China por estar al tanto de la evolución de mi tesis.

A todas las personas de mi barrio, por estar siempre atentos y preocupados por mi tesis.

A la instructora Esperanza por el apoyo que me brindó durante toda la carrera.

Al tribunal por todas sus recomendaciones.

A todos los profesores que me formaron como profesional.

A mis compañeros del 3501 en especial a Paúl, Jennifer, Arian, Alían, Leonardo, Alejandro Mariño y Osbel.

Yoel:

Agradecer a mi tutora Yalice Gámez Batista por toda la ayuda que nos brindó, por ser tan buena persona y por ayudarme a culminar mi carrera como profesional.

A mi compañero de tesis Manuel Arias Furones por ser mi fiel compañero, por ser una persona tan especial para mí, por su compañía estos años, por ser el mejor compañero de tesis del mundo, por llegar a ser como un hermano para mí.

A mis padres, abuelos, mi hermana y mi novia por todo lo que significan para mí.

A mis amigos de la universidad Carlos, Pedro, Keyler, Leo, Andrés, Joenis.

A mis amigos Adrián y Luis Ernesto.

A mis compañeros del 3502 por brindarme su amistad todos estos años.

RESUMEN

El hombre, tardó miles de años en desarrollar las bases de las matemáticas modernas, y miles más para llegar al desarrollo tecnológico en el área que se conoce en la actualidad como la microelectrónica. Sin embargo, en el siglo XX, la curva de crecimiento de estas tecnologías cambió significativamente, de tal manera, que en unos cuantos años se llegó a contar con impresionantes máquinas de cómputo.

Esto ha llevado a las sociedades modernas a cambiar por completo su mentalidad con respecto a la utilización de la herramienta más difundida en el mundo: la computadora. Es por eso que se utiliza como un medio para el cambio en los métodos de enseñanza y aprendizaje fundamentalmente en las universidades, con el objetivo de mejorar sus asignaturas, entre ellas las Matemáticas, para que sean más interactivas y garanticen una mejor asimilación de los conocimientos por parte de los estudiantes.

En la Universidad de las Ciencias Informáticas (UCI) actualmente no se cuenta con una herramienta diseñada en función de las características de los estudiantes de 1er año; y que sirva de apoyo en el proceso de enseñanza- aprendizaje de la asignatura de Matemática Discreta II, enfocado al tema de Relaciones de Recurrencia. Por estas razones este trabajo propone un sistema para el apoyo en la enseñanza de las relaciones de recurrencia, de forma tal que responda a la notación de la asignatura, a las características propias de los estudiantes del primer año de la carrera de Ingeniería en Ciencias Informáticas, y de forma general al proceso de enseñanza- aprendizaje de este tema en la asignatura de Matemática Discreta II.

Palabras Clave: Relaciones de recurrencia, software educativo, Asistente Matemático.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Las tecnologías de información y comunicación (TIC) han transformando la sociedad en sus esferas personal y profesional, cambiando las formas de acceso al conocimiento, aprendizaje, los modos de comunicación y la manera de establecer relaciones. Uno de los procesos en los que han tenido mayor impacto es en el de enseñanza- aprendizaje (PEA) con la incorporación de herramientas educativas, que facilitan y enriquecen el trabajo de los profesores y estudiantes a partir de una mayor interacción.

La Universidad de las Ciencias Informáticas (UCI) tiene como objetivo principal la formación de profesionales competentes y comprometidos con la Revolución en la rama de la informática. En esta universidad existe una facultad dedicada al primer año de la especialidad, Facultad Introductoria de Ciencias Informáticas (FICI), concebida en función de garantizar una mejor atención a los estudiantes en esta etapa de tránsito en sus niveles educacionales. Estos estudiantes son provenientes de todo el país por lo que se caracterizan por su gran heterogeneidad y desigualdad de origen, de nivel de aprovechamiento y de fuentes de motivación. Es por ello que supone un reto lograr el desarrollo de habilidades profesionales y el cumplimiento de los objetivos instructivos y educativos.

Como parte del Plan de Estudios de Ingeniería en Ciencias Informáticas, los estudiantes reciben la asignatura de Matemática Discreta II que es de vital importancia en su formación como informáticos por las bases teóricas que provee y las habilidades de pensamiento lógico y algorítmico que desarrolla. Todas las ciencias usan la matemática como instrumento para medir, calcular, estimar, definir, e incluso como lenguaje de expresión. Sin embargo, esta asignatura es una de las que ofrece mayor grado de dificultad para ellos por su complejidad y por el cambio de filosofía en el pensamiento que representa con respecto a sus conocimientos de enseñanzas precedentes.

La utilización de la computación en la enseñanza de las matemáticas, tanto con fines didácticos o como asistente matemático, tiene como objetivo incuestionable: capacitar a los alumnos para enfrentarse con adecuada formación a sus futuras actividades profesionales, así como favorecer la necesaria adaptación a los continuos avances de la tecnología (Johnsonbaugh, 2003).

Atendiendo a las características de los estudiantes de primer año, las potencialidades que brindan las herramientas matemáticas, y la complejidad que supone para ellos el aprendizaje de los contenidos de la Matemática Discreta II; el colectivo de profesores de esta asignatura

en la FICI se dio a la tarea de desarrollar un conjunto de herramientas de apoyo a la docencia que favorecieran una mejor comprensión de estos contenidos por parte de los estudiantes, teniendo en cuenta sus limitadas habilidades informáticas y las dificultades en el aprendizaje de la asignatura. De ellas falta por desarrollar la relacionada con el cálculo de relaciones de recurrencia. Ante tal circunstancia se identifica el siguiente **problema a resolver:**

¿Cómo desarrollar un software educativo para la enseñanza de las relaciones de recurrencia en la asignatura de la Matemática Discreta II que utilice la notación estándar de la asignatura y que muestre el procedimiento de solución?

Se precisa como **objeto de estudio** de este trabajo el proceso de desarrollo de software, y los conceptos fundamentales que lo sustentan se circunscriben al proceso de desarrollo de software educativo, siendo este su **campo de acción**.

En correspondencia con el problema identificado y el objeto a investigar, se propone como **objetivo general** desarrollar un sistema informático para el cálculo de relaciones de recurrencia que cumpla con la notación estándar de la asignatura y muestre el procedimiento de solución para la asignatura de Matemática Discreta II de la carrera de Ingeniería en Ciencias Informáticas.

Para darle solución al objetivo planteado se trazaron los siguientes **objetivos específicos**:

1-Elaborar el marco teórico de la investigación para lograr la correcta selección de la metodología y las tecnologías a utilizar en el desarrollo de la solución.

2-Diseñar un sistema para el cálculo de relaciones de recurrencias.

3-Implementar el sistema para el cálculo de relaciones de recurrencias con las características definidas en los procesos de análisis y diseño.

4-Validar los resultados obtenidos con la utilización de la aplicación informática desarrollada mediante la realización de un diseño experimental.

Para la realización de los objetivos anteriores se han concebido las siguientes **tareas de investigación**:

1. Construcción del marco teórico relacionado con los conceptos fundamentales de los softwares educativos, asistentes matemáticos y el cálculo de relaciones de recurrencia.
2. Análisis de herramientas, algoritmos y métodos empleados en los asistentes matemáticos para el cálculo de relaciones de recurrencia.
3. Realización de una entrevista que permita recopilar datos sobre las necesidades para la impartición del tema de relaciones de recurrencia a los estudiantes del primer año.

4. Identificar las funcionalidades implementadas en los asistentes matemáticos, sobre el tratamiento de las relaciones de recurrencia.
5. Selección de las herramientas, metodología y tecnologías adecuadas para el desarrollo de la solución.
6. Implementación de la solución teniendo en cuenta las herramientas, metodología y tecnologías seleccionadas.
7. Realización de pruebas de software a la solución desarrollada.
8. Aplicación práctica de la solución.

La investigación realizada se sustenta en la siguiente **idea a defender:** un sistema para el cálculo de relaciones de recurrencia que cumpla con la notación estándar de la asignatura y muestre el procedimiento de solución, propiciará la impartición de este tema a los estudiantes de primer año de la carrera de Ingeniería en Ciencias Informáticas en la asignatura de Matemática Discreta II.

Posibles resultados:

- Listado de las funcionalidades de un conjunto de asistentes matemáticos sobre el tratamiento de las relaciones de recurrencia.
- Sistema informático para el cálculo de las relaciones de recurrencia en la asignatura Matemática Discreta II, que utilice la notación estándar de la asignatura y muestre el procedimiento paso a paso, a través de los métodos de solución.

Para la realización de las tareas propuestas se utilizaron los siguientes métodos científicos de investigación:

Métodos Teóricos:

- **Histórico-lógico:** A través de este método se establece la necesaria correspondencia entre los elementos de los métodos lógico e histórico, proyectando el análisis de la evolución histórica de los fenómenos, con la proyección lógica de su comportamiento futuro (González Ramírez, 2006). Se refleja este método pues posibilita realizar un estudio sobre los antecedentes de los asistentes matemáticos, facilitando el análisis de la trayectoria de estos sistemas para una mejor comprensión de los mismos, realizar el análisis de la evolución, caracterización y

revelar las insuficiencias que se manifiestan en el cálculo de las relaciones de recurrencia.

- **Analítico-sintético:** Está integrado por el desarrollo del análisis y la síntesis, mediante el cual se descompone un objeto, fenómeno o proceso en los principales elementos que lo integran para analizar, valorar y conocer sus particularidades, y simultáneamente a través de la síntesis, se integran como un todo (González Ramírez, 2006). A partir del empleo de este método se realizará el análisis de las bases teóricas que sustentan el software educativo, de los asistentes matemáticos y del tratamiento que realizan del cálculo de relaciones de recurrencia. Esto permitirá analizar, valorar y conocer sus particularidades para la identificación de los requisitos de la solución a implementar.
- **Modelación**: Método mediante el cual se crean abstracciones con vistas a explicar la realidad. El modelo como sustituto del objeto de investigación. En el modelo se revela la unidad de lo objetivo y lo subjetivo (González Ramírez, 2006). Este método se refleja a la hora de realizar los modelos correspondientes al ciclo de vida del desarrollo de la herramienta propuesta. Propiciará la representación o modelo del sistema, lo cual permite el diseño de tarjetas CRC para la solución propuesta.

Métodos empíricos:

- **Tormenta de ideas:** La tormenta de ideas es la práctica que permite al investigador obtener información de primera mano, es una herramienta de trabajo grupal que facilita el surgimiento de nuevas ideas sobre un tema o problema determinado. La lluvia de ideas es una técnica de grupo para generar ideas originales en un ambiente relajado (González Ramírez, 2006). Se pone de manifiesto mediante conversaciones planificadas con los clientes, para obtener información acerca del problema en cuestión. Su uso constituye un medio para el conocimiento cualitativo de las características particulares de las funcionalidades de la herramienta. Es de gran beneficio para acumular opiniones de los profesores de Matemática Discreta II en cuanto a las deficiencias observadas en los asistentes matemáticos utilizados en la UCI.

El presente trabajo está estructurado de la siguiente manera:

Capítulo #1: Fundamentación teórica

Este capítulo contiene las cuestiones teóricas necesarias para la comprensión de la investigación, lo cual incluye el establecimiento del estado del arte a través del estudio de las características y funciones de los softwares educativos, los asistentes matemáticos más utilizados en la universidad, y el tratamiento al cálculo de relaciones de recurrencia que realizan. Se realiza un análisis y selección de la metodología de desarrollo en función de las necesidades del sistema y las características del equipo de desarrollo, así como el estudio de las herramientas y tecnologías a emplear en el desarrollo de la solución.

Capítulo #2: Análisis y diseño del sistema para el cálculo de relaciones de recurrencia en la asignatura de Matemática Discreta II

En este capítulo se realiza una descripción de la propuesta del sistema y sus principales funcionalidades. Se expone todo el diseño de la aplicación con los artefactos definidos por la metodología para esta fase.

Capítulo #3: Implementación y prueba del sistema para el cálculo de relaciones de recurrencia en la asignatura de Matemática Discreta II

En este capítulo se elabora la solución propuesta evidenciándose las funcionalidades del sistema para el cálculo de relaciones de recurrencia en la asignatura de Matemática Discreta II. Se muestran los resultados de la validación de los requisitos y el diseño, a través de la aplicación de técnicas. Además, se hace un resumen de los resultados obtenidos tras la aplicación de las pruebas de funcionamiento y unitarias realizadas al sistema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se analizan los principales conceptos asociados al software educativo, con el fin de servir de soporte teórico al desarrollo del sistema a realizar. Además, se realiza un análisis de los asistentes matemáticos más utilizados en la enseñanza de las Matemáticas en la Universidad de las Ciencias Informáticas. De igual forma se puntualizan las principales características y funcionalidades de las tecnologías a emplear en el desarrollo de la solución propuesta.

1.1. Conceptualización, importancia y funciones del software educativo

El desarrollo acelerado de la ciencia y la tecnología en el campo de la Informática y las Comunicaciones ha provocado que en la actualidad se implementen aplicaciones informáticas en todos los sectores de la sociedad. Específicamente, en el ámbito académico, se ha impulsado el uso y desarrollo del software como medio para potenciar la calidad del PEA de las futuras generaciones. El software educativo posibilita la transmisión de conocimientos de una forma más amena, integradora, diferenciada, reguladora y activa que el resto de los medios de enseñanza (Labrada, 2011).

Un medio de enseñanza tiene como misión fundamental facilitar el aprendizaje de los alumnos. En unos casos como refuerzo de la acción del profesor en clase y en situaciones, de forma presenciales, facilitando y mejorando la comunicación con los alumnos. En otros, también seleccionados y controlados por el profesor, se pueden mostrar autosuficientes para la explicación de un contenido (Ramos, 2004).

De acuerdo con (Sanchez, 2013), diversos autores de las ciencias pedagógicas han abordado desde sus investigaciones el concepto de software educativo:

- (Sanchez, 1999) define el concepto genérico de software educativo como cualquier programa computacional cuyas características estructurales y funcionales sirvan de apoyo al proceso de enseñar, aprender y administrar.
- (Lamas, 2000) lo precisa como una aplicación informática que, soportado sobre una bien definida estrategia pedagógica, apoya directamente el PEA constituyendo un efectivo instrumento para el desarrollo educacional del hombre.
- (Rizo, 2005) lo detalla como una aplicación informática concebida especialmente como medio integrado al PEA.

- (Labrada, 2011) lo enuncia como un medio didáctico digital autónomo, elaborado por un equipo multidisciplinario, encaminado al desarrollo de la personalidad de los educandos desde el punto de vista afectivo y cognitivo a partir de la integración de recursos y en correspondencia con los objetivos del currículo de la enseñanza y los destinatarios a que está dirigido.

La mayoría de los autores coinciden en las definiciones aportadas, el carácter instrumental del software educativo, a la vez que lo refieren como cualquier programa de computador creado específicamente para apoyar y ser utilizado en el PEA. La importancia de su desarrollo según (Sanchez, Toctaquiza, Marcelo, 2013) radica en las siguientes afirmaciones:

- Son altamente interactivos, a partir del empleo de recursos, como videos, sonidos, fotografías, hipertextos, diccionarios especializados, ejercicios y juegos instructivos que apoyan las funciones de evaluación y diagnóstico, permitiendo contestar inmediatamente las acciones de los estudiantes a través de un diálogo mutuo entre estos y el ordenador.
- Flexibilidad en su uso, tanto físico como de horario. El estudiante es capaz de decidir cuándo y cómo desea usarlo.
- Individualizan el trabajo; se adaptan al ritmo de trabajo de cada estudiante.
- Los estudiantes aprenden con rapidez y facilidad.
- Estimulan la construcción de conocimientos, nuevas formas de pensar, investigar y aprender haciendo.
- Desarrollan los procesos lógicos del pensamiento, la imaginación, la creatividad y la memoria.
- Facilitan las representaciones de procesos no perceptibles por el ojo humano en tiempo y espacio de forma animada.
- Simulan procesos complejos y optimizan el tiempo del que se dispone para impartir gran cantidad de conocimientos de forma amena y regulada por el usuario.
- Inciden en el desarrollo de habilidades psicomotrices y permiten retroalimentar al estudiante evaluando lo aprendido.

El desarrollo del software educativo tiene como base el poder desarrollar aplicaciones que soporten efectivamente el PEA. Es así, como el uso de las TIC abre nuevas posibilidades de innovación y realización de diferentes modelos pedagógicos que junto con la intrepidez,

curiosidad y motivación del maestro para con los estudiantes, tienda a mejorar y cambiar de una forma positiva el proceso educativo (Collaguazo, 2016).

La herramienta que se necesita como medio de enseñanza para la asignatura Matemática Discreta II en la UCI:

- Será concebida con un propósito específico: apoyar la labor del Departamento de Ciencias Básicas de la FICI en el proceso de aprendizaje de los estudiantes, por lo cual resulta necesario tener presente las características mencionadas anteriormente y definir cuáles incorporará.
- Optimizará el tiempo que dispone el profesor para impartir el sistema de conocimientos asociado al tema de relaciones de recurrencia.
- Mostrará la ejecución, paso a paso, de los métodos que implemente y de forma similar a como se estudian en la asignatura.

Los programas didácticos, como también se les conoce, cuando se aplican a la realidad educativa, realizan funciones básicas propias y además en algunos casos, en dependencia de la manera en que el profesor organice su utilización y aplique la misma en el salón de clases podrá desempeñar diferentes funciones. Sin embargo, como ocurre con otros sistemas educativos, no se puede afirmar que el software educativo por sí mismo sea bueno o malo, todo dependerá del uso que de él se haga y de cómo se utilice en cada situación concreta. En última instancia, su funcionalidad, ventajas e inconvenientes que pueda reportar su uso, serán el resultado de las características del material, de su adecuación al contexto educativo al que se aplica y de la manera en que el profesor organice su utilización (García, 2011).

Resulta importante también estudiar las funciones que puede lograr un software educativo en los alumnos, según se describe en (Sanchez, Toctaquiza, Marcelo, 2013). Estas funciones son:

- Función informativa: Presentan contenidos que proporcionan información estructurada de la realidad a los estudiantes.
- Función instructiva: Promueven determinadas actuaciones en los estudiantes, las cuales están encaminadas a facilitar el logro de los objetivos educativos específicos.
- Función motivadora: Los estudiantes se sienten atraídos e interesados por todo el software educativo. Los programas suelen incluir elementos capaces de captar la atención de los alumnos, mantener su interés y cuando sea necesario, enfocarlo hacia los aspectos más importantes de las actividades.

- Función evaluadora: La interactividad propia de estos materiales, que les permite responder inmediatamente a las respuestas y acciones de los estudiantes, los hacen especialmente adecuados para evaluar el trabajo que se realice con ellos. Esta evaluación puede ser de dos tipos:
    a. Implícita, cuando el estudiante detecta sus errores, se evalúa, a partir de las respuestas que le da el ordenador.
    b. Explícita, cuando el programa presenta informes valorando la actuación del alumno. Este tipo de evaluación sólo lo realizan los programas que disponen de módulos específicos de evaluación.
- Función investigadora: Ofrecen a los estudiantes interesantes entornos donde investigar y buscar determinadas informaciones.
- Función innovadora: Aunque no siempre los planteamientos pedagógicos que proponen resulten innovadores, el software educativo se puede considerar material didáctico con esta función, pues utiliza tecnología recientemente incorporada a los centros educativos y, en general, suele permitir muy diversas formas de uso. Esta versatilidad abre amplias posibilidades de experimentación didáctica e innovación educativa en el aula.
- Función lúdica: El software educativo tiene como finalidad reforzar de manera lúdica el conocimiento a través de juegos interactivos donde el estudiante adquiere las habilidades propuestas de una manera divertida.

De las funciones anteriores, atendiendo a las necesidades de los profesores de la asignatura Matemática Discreta II en la FICI, la solución propuesta deberá cumplir con las siguientes:

- Función instructiva: el software deberá ser capaz de guiar al estudiante durante su interacción con la aplicación a través de una correcta estructuración de menús, con la finalidad de que el alumno adquiera los conocimientos necesarios y pueda utilizarlos para cumplir los objetivos de la asignatura.
- Función informativa: representará de forma estructurada y ordenada los contenidos que se imparten en la asignatura Matemática Discreta II. Además, estos contenidos deben estar presentados de manera similar a como los estudiantes los reciben en las aulas
- Función motivadora: la forma de organizar y representar el contenido deber ser amigable para los alumnos, la información y resultados de los métodos deben resaltar

por encima de lo demás, brindándole importancia a los elementos significativos en los cálculos y operaciones.

Luego de los análisis realizados hasta el momento, y determinadas las características y principales funciones que debe cumplir la propuesta de solución, resulta de vital importancia analizar el sistema de conocimientos al que va dirigida, de forma que responda funcionalmente a las necesidades de los clientes.

1.2. Relación de recurrencia

Conceptos tales como sucesión, polinomio característico, condición inicial, coeficientes indeterminados e iterativos, entre otros, propios del campo que ocupan, están presentes en todas las ramas del saber matemático. La noción de las relaciones de recurrencia es básica en las matemáticas y en consecuencia, se sitúa en los fundamentos de estas.

Una relación de recurrencia para una sucesión {a0; a1; a2; a3...} es una fórmula que expresa cada término a_n, a partir de cierto $n_0 \in N$, en función de uno o más de los términos que le preceden. Los valores de los términos necesarios para empezar a calcular se llaman condiciones iniciales. Se dice que una sucesión es una solución de la relación de recurrencia si su término general verifica dicha relación. Estas relaciones están estructuradas por relaciones de recurrencia **lineales** y **no lineales**, estas primeras presentan dos tipos de clasificaciones las **homogéneas y las no homogéneas** (Johnsonbaugh, 2003).

Una relación de recurrencia para la sucesión a_0, a_1,… es una ecuación que relaciona a_n con algunos de sus predecesores a_0 a_1,… a_{n-1}.

Condiciones Iniciales:

Las condiciones iniciales para la sucesión a_0 a_1,…son valores dados en forma explícita para un número finito de términos de la sucesión.

Ejemplo:

La sucesión de Fibonacci se define mediante la relación de recurrencia $f_n=f_{n-1}+f_{n-2}$, para n>=3 y las condiciones iniciales $f_1 =1$, $f_2=2$ (Johnsonbaugh, 2003).

1.2.1. Relaciones de recurrencia lineales homogéneas:

Definición: Se nombra ecuación característica de la relación de recurrencia $a_n = c_1a_{n-1} + c_2a_{n-2}$ a la ecuación $r^2 - c_1r - c_2 = 0$. A sus raíces se les llama raíces características. Si $a_n = c_1a_{n-1} + c_2a_{n-2} \cdots + c_ma_{n-m}$, para $n \geq m$, se dice que la relación de recurrencia es lineal homogénea de orden m (Johnsonbaugh, 2003).

Teorema: Dada la relación de recurrencia $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ con $c_1 6= 0\ 6= c_2$:

1) α es raíz característica si y solo si $a_n = \alpha^n$ es solución de la relación de recurrencia,

2) si α es raíz doble de la ecuación característica, entonces $a_n = n\alpha^n$ es solución de la relación de recurrencia,

3) si T y S son soluciones de la relación de recurrencia, entonces S + T y kS también lo son, para todo $k \in R$.

4) Si la ecuación $x^2 - c_1 x - c_2 = 0$ tiene dos soluciones reales distintas α y β se tiene que $a_n = C_1\alpha^n + C_2\beta^n$.

5) Si la ecuación $x^2 - c_1 x - c_2 = 0$ tiene una solución real doble α se tiene que $a_n = (C_1 + C_2 n)\ \alpha^n$.

C_1 y C_2 se determinan a partir de las condiciones iniciales a_0 y a_1 (Calderón, 2000).

Ejemplo: Para la sucesión de Fibonacci, aplicando el teorema anterior se obtiene que:

$$a_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

Observación: Todo se generaliza a relaciones de recurrencia del tipo $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots c_k a_{n-k}$ (Johnsonbaugh, 2003).

Algoritmo de solución para las relaciones de recurrencia lineales homogéneas: Dada una relación de recurrencia, se determina la linealidad. De ser lineal, homogénea y de coeficientes constantes, se aplica el método de solución del Polinomio característico (Calderón, 2000):

1) Determinar el polinomio y la ecuación característica.

2) Resolver la ecuación característica.

3) Determinar las raíces características.

5) Expresar la solución.

6) Resolver el sistema de ecuaciones lineales, sustituyendo los valores iniciales.

7) Expresar la solución cerrada.

1.2.2. Relaciones de recurrencia lineales no homogéneas:

Definición: Si $a_n = c_1 a_{n-1} + c_2 a_{n-2} \cdots + c_m a_{n-m} + g(n)$, para $n \geq m$, se dice que la relación de recurrencia es lineal no homogénea de orden m. A la relación $a_n = c_1 a_{n-1} + c_2 a_{n-2} \cdots + c_m a_{n-m}$, resultante de eliminar g(n) se le llama relación de recurrencia lineal homogénea asociada.

Teorema: Una solución particular (x_n) de la relación de recurrencia lineal no homogénea se puede encontrar en algunos casos especiales (Calderón, 2000):

- Si $g(n) = P_k(n)$ (polinomio de grado k), entonces $x_n = Q_k(n)$ (polinomio de grado k), excepto si 1 es raíz característica con multiplicidad s, en cuyo caso $x_n = n^s Q_k(n)$.
- Si $g(n) = pa^n$, $p \in R$, entonces $x_n = qa^n$, $q \in R$, excepto si a es raíz característica con multiplicidad s, en cuyo caso $x_n = qn^s a^n$.
- Si $g(n) = a^n P_k(n)$, entonces $x_n = a^n Q_k(n)$, excepto si a es raíz característica con multiplicidad s, en cuyo caso $x_n = n^s a^n Q_k(n)$.

Algoritmo de solución para las relaciones de recurrencia lineales no homogéneas: Dada una relación de recurrencia, se determina la linealidad, de ser no lineal, homogénea y de coeficientes constantes se aplica el método de solución de Coeficientes Constante (Calderón, 2000):

1) Se obtiene la solución general de la ecuación homogénea asociada.
2) Se obtiene una solución particular de la relación de recurrencia no homogénea.
3) La suma de la solución general de la ecuación lineal homogénea asociada y de una solución particular de la relación de recurrencia lineal no homogénea da la solución general de la relación de recurrencia lineal no homogénea.
4) La solución específica se obtiene a partir de las condiciones iniciales.

1.3. Principales asistentes matemáticos

Los **asistentes matemáticos** (programas diseñados con intencionalidad pedagógica) son recursos tecnológicos que permiten al estudiante concentrar esfuerzos en el razonar, solucionar y formular problemas, así como en verificar teoremas y propiedades matemáticas (Trochez, 2013).

Aun cuando los asistentes matemáticos no constituyen softwares educativos, su análisis constituye un importante referente teórico en el marco de esta investigación. Al ser el contenido la categoría principal dentro de las categorías de la didáctica, el tratamiento a las relaciones de recurrencia por estos softwares especializados es de vital importancia para el desarrollo de la propuesta. Dentro de los asistentes matemáticos más utilizados en la Universidad de las Ciencias Informáticas se encuentran:

- **Mathematica** es un programa utilizado en áreas científicas, de ingeniería, matemáticas y áreas computacionales. Originalmente fue concebido por Stephen Wólfram, quien

continúa siendo el líder del grupo de matemáticos y programadores que desarrollan el producto en Wólfram Research. Casi cualquier flujo de trabajo incluye resultados de computación, y eso es lo que hace Mathematica: desde la construcción de un sitio web de comercio de fondos de cobertura o la publicación de libros de texto de ingeniería interactivos, desarrollo de algoritmos de reconocimiento de imágenes incrustadas o cálculo enseñanza. Es conocida como la aplicación final del mundo para sus cálculos. Pero es mucho más: es la única plataforma de desarrollo de la plena integración de la computación en flujos de trabajo completos, que se mueve sin problemas desde las ideas iniciales hasta el final al individuo desplegado o soluciones empresariales (Trochez, 2013). Este asistente matemático contiene las siguientes funcionalidades (Calderón, 2000):

- Aplica el método iterativo, a través de matrices.
- Resuelve sistema de ecuaciones lineales utilizando matrices, para determinar la solución general o particular en caso de introducir valores iniciales.
- Aplica los métodos del polinomio característico y el de los coeficientes indeterminados.
- Resuelve polinomios, a través de comandos para determinar los valores de las raíces.

Sin embargo, en el tema de relaciones de recurrencia este asistente no permite realizar las siguientes funcionalidades:

- Generar relación de recurrencia según la linealidad.
- Generar relación de recurrencia según la homogeneidad.
- Generar relación de recurrencia según la naturaleza de los coeficientes.
- Utilización de la notación usual de las relaciones de recurrencia.
- Generar relación de recurrencia según la naturaleza de los coeficientes.
- Determinar la linealidad de una relación de recurrencia.
- Determinar la homogeneidad de una relación de recurrencia.
- Determinar la naturaleza de los coeficientes de una relación de recurrencia.
- Determinar el orden de una relación de recurrencia.

- **MATLAB** es un programa para realizar cálculos numéricos con vectores y matrices. Como caso particular puede también trabajar con números escalares tanto reales como complejos, con cadenas de caracteres y con otras estructuras de información más complejas. Una de las capacidades más atractivas es la de realizar una amplia variedad de gráficos en dos y tres dimensiones. Tiene también un lenguaje de programación propio. Es un gran programa de cálculo técnico y científico. MATLAB es un lenguaje de alto rendimiento para la informática técnica. Integra cómputo, visualización y programación en un formato fácil de usar, en un entorno donde los problemas y soluciones se expresan familiarizados en notación matemática. Este asistente matemático realiza las funcionalidades siguientes (García de Jalón, 2000):
    - Aplicar el método del polinomio característico.
    - Resolver polinomios a través de comandos, introduciendo los valores de los coeficientes del polinomio.
    - Determinar los valores de las raíces características.
    - Resolver el sistema de ecuaciones lineales, que permite determinar la solución general o particular.
    - Algoritmo de desarrollo.
    - Matemáticas y computación.
    - Modelado, simulación y prototipo.
    - Análisis de datos, exploración y visualización.
    - Ciencia y la Ingeniería Gráfica.
    - Desarrollo de aplicaciones, incluyendo la construcción de interfaces gráficas de usuario.

  Dicho asistente no permite realizar las siguientes funcionalidades sobre las relaciones de recurrencia:

    - Generar relación de recurrencia según la linealidad.
    - Generar relación de recurrencia según la homogeneidad.
    - Generar relación de recurrencia según la naturaleza de los coeficientes.
    - Utilización de la notación usual de las relaciones de recurrencia.
    - Generar relación de recurrencia según la naturaleza de los coeficientes.
    - Determinar la linealidad de una relación de recurrencia.

- Determinar la homogeneidad de una relación de recurrencia.
- Determinar la naturaleza de los coeficientes de una relación de recurrencia.
- Determinar el orden de una relación de recurrencia.
- Resolver la relación de recurrencia a través de los métodos coeficiente indeterminado e iterativo.

❖ **Derive** es un programa de álgebra computacional desarrollado por Texas Instruments. Es un potente programa para el cálculo matemático avanzado: variables, expresiones algebraicas, ecuaciones, funciones, vectores, matrices, trigonometría, entre otros. Derive es un poderoso sistema para hacer operaciones matemáticas simbólicas y numéricas. Procesa variables algebraicas, expresiones, ecuaciones, funciones, vectores, matrices y expresiones booleanas como una calculadora de procesos científicos. Los problemas en los campos de la aritmética, álgebra, trigonometría, cálculo, álgebra lineal y cálculo proposicional se pueden resolver con un clic del ratón. Grafica expresiones matemáticas en dos y tres dimensiones utilizando diferentes sistemas de coordenadas. Por su perfecta integración de las capacidades numéricas, algebraicas y gráficas, Derive es una excelente herramienta y uno de los mejores asistentes matemáticos (Fernández Álvarez, 2004).

Este asistente matemático permite realizar las siguientes funcionalidades (Calderón, 2000):

- Cálculos simbólicos.
- Trabajar con constantes simbólicas.
- Resolver polinomios.
- Resolver sistemas de ecuaciones.

En el tema de relaciones de recurrencia este asistente no permite realizar las siguientes funcionalidades:

- Generar relación de recurrencia según la linealidad.
- Generar relación de recurrencia según la homogeneidad.
- Generar relación de recurrencia según la naturaleza de los coeficientes.
- Utilización de la notación usual de las relaciones de recurrencia.
- Generar relación de recurrencia según la naturaleza de los coeficientes.
- Determinar la linealidad de una relación de recurrencia.
- Determinar la homogeneidad de una relación de recurrencia.

- Determinar la naturaleza de los coeficientes de una relación de recurrencia.
- Determinar el orden de una relación de recurrencia.
- Resolver la relación de recurrencia a través de los métodos coeficiente indeterminado e iterativo.
- Expresar la solución general.

❖ **Maple**: Programa desarrollado en la década de 1980 por el grupo de Cálculo simbólico de la Universidad de Waterloo, Canadá. El mismo está orientado a la computación técnica de ingenieros, matemáticos y científicos con el propósito de que estos profesionales realicen cálculos rápidos, simbólicos y algebraicos en pro de la resolución de problemas matemáticos, el desarrollo de hojas de trabajo interactivas y modelos de simulación en 2D y 3D, entre otros. Se caracteriza por (Hung, 2014):

- Completo soporte de documentación de gráficos, el cual le permitirá agregar información adicional a sus gráficos 3D, haciéndolos más atractivos y mucho más fáciles de interpretar. Las anotaciones disponibles incluyen títulos y etiquetas que contienen texto, expresiones matemáticas, flechas, dibujos a mano alzada, figuras geométricas, etiquetas de múltiplos de pi (π) y ejes coloreados individualmente.
- Al utilizar controles de rotación completa, puede cambiar interactivamente la visualización utilizando los tres ángulos de rotación, lo que le permite enfocar los puntos de interés del gráfico con facilidad.
- Puede crear gráficos 3D y 2D que involucren unidades. Estas unidades serán agregadas automáticamente como información adicional a los ejes de sus gráficos.

Este asistente matemático permite realizar las siguientes funcionalidades:

- Resuelve polinomios, utilizando comandos nops.
- Resuelve relaciones de recurrencia aplicando los métodos del polinomio característico y coeficientes indeterminados.
- Resuelve sistemas de ecuaciones lineales.

Sin embargo, en el tema de relaciones de recurrencia este asistente no permite realizar las siguientes funcionalidades:

- Generar relación de recurrencia según la linealidad.
- Generar relación de recurrencia según la homogeneidad.

- Generar relación de recurrencia según la naturaleza de los coeficientes.
- Utilización de la notación usual de las relaciones de recurrencia.
- Generar relación de recurrencia según la naturaleza de los coeficientes.
- Determinar la linealidad de una relación de recurrencia.
- Determinar la homogeneidad de una relación de recurrencia.
- Determinar la naturaleza de los coeficientes de una relación de recurrencia.
- Determinar el orden de una relación de recurrencia.
- Resolver la relación de recurrencia a través del método iterativo.
- Expresar la solución general.

A partir del análisis realizado de los asistentes matemáticos actualmente utilizados en la universidad, se pudo constatar que los mismos no incorporan todas las funcionalidades necesarias en la asignatura de Matemática Discreta II para el trabajo con las relaciones de recurrencia. De forma general incluyen operaciones que pueden ser utilizadas pero que hacen muy engorroso el procedimiento de solución. Por ejemplo:

- El MATLAB es un asistente con un nivel de complejidad bastante elevado para un estudiante de 1er año, ya que implementa sus operaciones a través de llamadas a funciones utilizando su propio lenguaje de programación. No utiliza la notación estándar establecida por la asignatura y requiere de un elevado nivel de abstracción en los estudiantes que dificulta su aprendizaje. Es por ello que se descarta como solución para el problema planteado.
- El Derive es el más adecuado y con más facilidad de uso de los asistentes mencionados anteriormente para un estudiante. Usa la notación establecida en la asignatura para la representación de las relaciones de recurrencia, pero de modo general no cuenta con todas las funcionalidades necesarias para el trabajo con el tema. Es por ello que se concluye que tampoco da respuesta al problema planteado.
- El asistente Maple tiene como ventajas para el trabajo con las relaciones de recurrencia que trabaja con polinomios característicos a través de comandos. Resuelve sistemas de ecuaciones lineales, aplica los métodos de solución del polinomio característico y de los coeficientes indeterminados, pero tiene como deficiencia que no puede clasificar una relación de recurrencia según los criterios de linealidad, homogeneidad, naturaleza de los coeficientes y el orden. No utiliza la

notación de la asignatura y realiza llamadas a funciones que un estudiante de primer año no domina. Por estas razones aplicar un método de solución se hace complejo. De esta forma se llega a la conclusión de que este asistente matemático tampoco constituye una respuesta al problema.

- El Mathematica es un intermedio en cuanto a complejidad entre el MATLAB y el Derive. Dicho asistente utiliza la notación que demanda la asignatura para el trabajo con las relaciones de recurrencia, pero trabaja a través de llamadas a funciones que resultan complejas para un estudiante de 1er año. Es por ello que también se descarta como solución al problema.

Para determinar cuál de ellos se ajusta en mayor medida al problema planteado con el apoyo de los autores (García de Jalón, 2001), (Hung, 2014), (Calderón, 2000), se identificaron un conjunto de requisitos que deben cumplir y que se muestran a continuación (ver Tabla 1). Atendiendo a la política establecida por el país y la universidad el asistente debe ser libre de costo en primer lugar; y como requisito del cliente debe ser un software multiplataforma, que trabaje las relaciones de recurrencia de forma compatible con las indicaciones metodológicas de la asignatura de Matemática Discreta II. Debe ser de fácil uso para estudiantes de primer año con incipientes habilidades informáticas, y debe además contener funcionalidades específicas para el trabajo de relaciones de recurrencia.

Tabla 1.Comparación entre los asistentes matemáticos (elaboración propia)

Asistente Matemático	Software Libre	Multiplataforma	Notación de la asignatura	Facilidad de uso	Aborda las relaciones de recurrencia	Incorpora todas funcionalidades
Mathematica	**No**	**Sí**	**Sí**	**Sí**	**Sí**	**No**
Matlab	**No**	**Sí**	**No**	**No**	**Sí**	**No**
Derive	**No**	**Sí**	**Sí**	**Sí**	**Sí**	**No**
Maple	**No**	**Sí**	**No**	**No**	**Sí**	**No**

Como se observa en la Tabla 1, de forma general ninguno de los asistentes incluye todos los requisitos necesarios para dar respuesta al problema. Los que más requisitos abarcan son el Mathematica y el Derive sin embargo, son softwares propietarios y no implementan todas las funcionalidades especificas demandadas por el cliente. De esta forma se confirma la

necesidad del desarrollo de un sistema informático que sí cumpla con todas estas características.

1.4. Metodología, herramientas y tecnologías seleccionadas para el desarrollo de la solución

La elección de la metodología, lenguajes y herramientas de desarrollo es un proceso de gran importancia, ya que de ello depende la calidad del desarrollo y del producto final. Para la realización de este proceso se tuvieron en cuenta las características del equipo de desarrollo, así como las del sistema que se desea desarrollar.

Las necesidades y características existentes del equipo de desarrollo a tener en cuenta para la propuesta de solución, son:

- Se cuenta con un equipo de desarrollo pequeño, del cual el cliente es miembro.
- Es más importante crear un producto software que funcione que escribir documentación exhaustiva.
- El tiempo de desarrollo es corto (máximo 5 meses).

Los parámetros establecidos considerados para el desarrollo de la propuesta de solución y que contribuyeron al análisis de las necesidades y especificaciones del problema planteado, son los siguientes:

- Dimensión de los datos conocida y limitada
- Aplicación de escritorio
- Portable
- Sin restricciones de acceso
- La complejidad del sistema está en el algoritmo que clasifica las relaciones de recurrencia
- Debe ser tecnológicamente compatible con las otras herramientas ya desarrolladas para la asignatura.

Además, los elementos que se obtienen de la experiencia del usuario nutren al autor de mecanismos informáticos para desarrollar una aplicación de escritorio utilizando la metodología, las herramientas y las tecnologías que se describen a continuación:

1.4.1. Metodología de desarrollo

La metodología de desarrollo de software guía el proceso de desarrollo para alcanzar la satisfacción del cliente y del equipo de desarrollo. Se debe seleccionar la metodología que

mejor se adapte al equipo de trabajo y al sistema a desarrollar. A continuación, se hace un análisis para la selección de la metodología de desarrollo de software.

Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo (Inteco, 2009).

Según la filosofía de desarrollo se pueden clasificar las metodologías en dos grupos: las metodologías tradicionales, que se basan en una fuerte planificación durante todo el desarrollo y las metodologías ágiles, en las que el desarrollo de software es incremental, cooperativo, sencillo y adaptado (Inteco, 2009).

Las metodologías tradicionales centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto, definido todo esto, en la fase inicial del desarrollo del proyecto. Se focalizan en la documentación, planificación y procesos (Inteco, 2009).

Las metodologías ágiles ponen de relevancia que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan (Inteco, 2009).

Se opta por la utilización de un enfoque ágil debido a que el tiempo para la realización del sistema es corto, el cliente está disponible a tiempo completo y el sistema a desarrollar es pequeño y cuenta con pocos requisitos funcionales.

Una vez seleccionado el enfoque, corresponde seleccionar la metodología. Para la selección de la metodología se analizan varios exponentes de la misma: Programación Extrema (XP), Scrum y Crystal. A continuación, se muestra una tabla comparativa entre estás metodologías (ver Tabla 2).

Tabla 2. Comparación entre las principales metodologías ágiles (Letelier, 2006)

Indicadores	XP	Scrum	Crystal
Participación del cliente	Activa, retroalimentación continua entre el cliente y el equipo de desarrollo.	En cada iteración	Activa
Pruebas	Ejecutadas constantemente ante cada modificación del sistema.	En cada iteración	Diariamente
Documentación	Se sustituye la documentación por la comunicación.	Poca	Poca
Centra la atención	Programadores	Gestión del proyecto	Programadores

Teniendo en cuenta la comparación anterior, se establece como metodología para guiar el desarrollo de la solución XP, por las siguientes razones:

- El cliente está en constante comunicación con el equipo de desarrollo.
- Las pruebas de unidad se realizan ante cada modificación del sistema, lo que garantiza el correcto funcionamiento de la solución.
- El sistema es pequeño y con pocos requisitos funcionales, por lo que no es necesario generar mucha documentación.
- El equipo de desarrollo posee conocimientos de la metodología.
- Es fácil de aplicar.

Programación Extrema:

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo. Se basa en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico (Inteco, 2009).

La metodología XP define como artefacto primordial las **historias de usuario** que son la base del software a desarrollar. Estas historias son escritas por el cliente y describen las interrelaciones entre los usuarios y el sistema y generalmente son complementadas con otro tipo de descripción. A partir de ellas y de la arquitectura que se utilizará, se planifican las entregas, así como los objetivos de cada una y las iteraciones con que contará (Alitimón, 2014).

Dentro de sus ventajas, se puede decir que la programación es más organizada, los programadores estarán satisfechos y la tasa de error disminuirá.

La Programación Extrema asume que la planificación nunca será perfecta y que los requisitos pueden cambiar a lo largo del ciclo de vida del software, a medida que varíen las necesidades del negocio.

El ciclo de vida ideal consta de cuatro etapas (Beck, 1999):

- **La planeación:** es la etapa inicial de todo proyecto en XP. En este punto se comienza a interactuar con el cliente y el resto del grupo de desarrollo para descubrir los requerimientos del sistema. En este punto se identifican el número y tamaño de las

iteraciones al igual que se plantean ajustes necesarios a la metodología según las características del proyecto. En este apartado se tendrán en cuenta ocho elementos, los cuales son los siguientes: Historias de usuario, velocidad del proyecto, iteraciones, entregas pequeñas, reuniones, roles en XP, traslado del personal y ajuste a XP.

- **Diseño:** en esta etapa se diseñan aquellas historias de usuario que el cliente ha seleccionado para la iteración actual por dos motivos: por un lado, se considera que no es posible tener un diseño completo del sistema y sin errores desde el principio. El segundo motivo es que, dada la naturaleza cambiante del proyecto, el hacer un diseño muy extenso en las fases iniciales del proyecto para luego modificarlo, se considera un desperdicio de tiempo.
- **La codificación:** es un proceso que se realiza en forma paralela con el diseño y la cual está sujeta a varias observaciones por parte de XP consideradas controversiales por algunos expertos tales como la rotación de los programadores o la programación en parejas.
- **Prueba:** XP enfatiza mucho los aspectos relacionados con las pruebas, clasificándolas en diferentes tipos y funcionalidades específicas, indicando quién, cuándo y cómo deben ser implementadas y ejecutadas. Del buen uso de las pruebas depende el éxito de otras prácticas, tales como la propiedad colectiva del código y la refactorización. Cuando se tienen bien implementadas las pruebas no habrá temor de modificar el código del otro programador en el sentido que, si se daña alguna sección, las pruebas mostrarán el error y permitirán encontrarlo.

XP se basa en la simplicidad, la comunicación y el reciclado continuo de código, para algunos no es más que aplicar una pura lógica. Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Los objetivos de XP son muy simples: la satisfacción del cliente. Esta metodología trata de dar al cliente el software que necesita y cuando lo necesita. Por tanto, se debe responder muy rápido a las necesidades del cliente, incluso cuando los cambios sean al final de ciclo de la programación. El segundo objetivo es potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo del software (Beck, 1999).

1.4.2. Paradigma de programación orientado a objetos

La programación orientada a objetos (POO u *OOP* según siglas en inglés) es un paradigma de programación que define los programas en términos de "clases de objetos". La programación orientada a objetos es una nueva forma de pensar acerca del proceso de descomposición de problemas y de desarrollo de soluciones de programación que brinda la posibilidad de definir clases a partir de otras clases ya construidas.

La programación orientada a objetos se ha convertido en el paradigma de programación más extendido en la actualidad. Sus conceptos y técnicas son ampliamente utilizados para reducir la complejidad del desarrollo de grandes sistemas de software. Este paradigma tuvo su origen dentro del mundo académico universitario y sigue evolucionando gracias a esfuerzos de investigación tanto de ámbito universitario como empresarial.

El paradigma orientado a objetos (OO) se refiere a un estilo de programación. La programación orientada a objetos (POO) es una manera de diseñar y desarrollar software que trata de imitar la realidad tomando algunos conceptos esenciales de ella; es decir modelar la realidad del problema a través de entidades independientes pero que interactúan entre sí y cuyas fronteras no permanezcan determinadas por su instrumentación computacional sino por la naturaleza del problema. Estas entidades serán denominadas objetos por analogía con el concepto de objeto en el mundo real (Katrib, 2003).

La POO es una extensión de los lenguajes de alto nivel estructurados que trata de representar de una forma más sencilla el modelo del mundo real. La POO intenta resolver principalmente los problemas de la Ingeniería del Software como: portabilidad, reusabilidad, mantenibilidad, entre otros. Para ello se base en características claves como el encapsulamiento, la herencia, el polimorfismo y el desarrollo orientado primero hacia el qué, y luego hacia el cómo (interfaces) (Meyer, 1999).

Está basado en los principios: herencia, abstracción, polimorfismo y encapsulamiento. Su uso se popularizó a principios de la década de los años 1990. En la actualidad, existe variedad de lenguajes de programación que soportan la orientación a objetos. Entre los conceptos básicos de la programación orientada a objetos se encuentran los objetos, las clases, los mensajes, la herencia, entre otros criterios básicos pertenecientes a este modelo de programación (Echeverría Rodríguez, 2006).

La programación orientada a objetos se utiliza en la creación del software, debido a las grandes ventajas que aporta como, por ejemplo (Echeverría Rodríguez, 2006):

- **Reusabilidad**: Cuando se han diseñado adecuadamente las clases, se pueden usar en distintas partes del programa y en numerosos proyectos.
- **Mantenibilidad**: Debido a la sencillez para abstraer el problema, los programas orientados a objetos son más sencillos de leer y comprender, pues permiten ocultar detalles de implementación dejando visibles sólo aquellos detalles más relevantes.
- **Modificabilidad**: La facilidad de añadir, suprimir o modificar nuevos objetos permite hacer modificaciones de una forma muy sencilla.
- **Fiabilidad**: Al dividir el problema en partes más pequeñas se pueden probar de manera independiente y aislar mucho más fácilmente los posibles errores que puedan surgir.

1.4.3. Lenguaje de programación

Un lenguaje de programación es un lenguaje artificial que se utiliza para expresar programas de ordenador. Está formado por un conjunto de símbolos, palabras claves utilizables y por reglas gramaticales para construir sentencias sintáctica y semánticamente correctas (Sala, 2015).

Existen múltiples lenguajes de programación como el C++, el C#, el Java entre otros. Se selecciona el lenguaje Java para el desarrollo de la solución ya que se ajusta con la descripción del sistema que se quiere desarrollar, es el lenguaje utilizado por las restantes aplicaciones desarrolladas por el colectivo de asignatura y de esta forma se garantiza la compatibilidad y además se tiene experiencia en su uso. Este lenguaje es adecuado para obtener un producto de pequeñas dimensiones, simple y portátil sobre diferentes plataformas y sistemas operativos.

La tecnología Java está compuesta básicamente por dos elementos: el lenguaje Java y su plataforma (plataforma se refiere a la máquina virtual de Java).

❖ **Lenguaje de programación Java**

Java es un lenguaje de programación orientado a objetos y actualmente es uno de los lenguajes más usados para la programación en todo el mundo. Es un lenguaje de alto nivel, que tiene la finalidad de obtener un producto de pequeñas dimensiones, simple y portátil sobre diferentes plataformas y sistemas operativos. Es un lenguaje de desarrollo de propósito general, y como tal es válido para realizar todo tipo de aplicaciones profesionales (García de Jalón, 2000).

La compañía Sun[1] describe el lenguaje Java como "simple, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico" (García de Jalón, 2000). Todas estas características son importantes, sin embargo, destacan tres, que son las que han proporcionado tanto interés por el lenguaje: la portabilidad, el hecho de que sea de arquitectura neutral y su simplicidad. Java ofrece toda la funcionalidad de los lenguajes potentes, pero sin las características menos usadas y más confusas de éstos.

- **Arquitectura neutral:** Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows Nt, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos.
- **Portabilidad:** Al ser de arquitectura neutral es altamente portable, es decir puede ser utilizado en cualquier computadora sin necesidad de ser instalado.
- **Simplicidad:** Basado en el lenguaje C++, pero donde se eliminan muchas de las características POO que se utilizan esporádicamente y que ocasionaban problemas frecuentes entre los programadores. La eliminación de causas de error y problemas de mantenimiento facilita y reduce el coste del desarrollo de software.

❖ **Máquina virtual de Java 8.0**

La Máquina Virtual de Java (Java Virtual Machine (JVM), por sus siglas en inglés) constituye un elemento imprescindible para la ejecución de un software desarrollado en esta tecnología. Es una aplicación que interpreta y ejecuta programas escritos en el lenguaje de programación Java. Específicamente puede interpretar el bytecode generado al compilar en Java. La JVM se encarga de terminar de compilar el bytecode en lenguaje máquina para que la aplicación Java pueda ser ejecutada en un dispositivo específico, este es el caso de las JVM que utilizan un compilador JIT (Just In Time) (Rojas, 2003).

1.4.4. Marcos de trabajo de Java

Los marcos de trabajo simplifican el desarrollo mediante la automatización de las tareas comunes. Así mismo, proporcionan estructura al código fuente, forzando al programador a crear código más legible y fácil de mantener. Es por ello que el uso de un marco de trabajo se hace indispensable en el desarrollo de software. Múltiples sitios web y aplicaciones son programados en Java y debe utilizarse una máquina virtual Java para poder ejecutarse, por lo

[1] Sun Microsystems: compañía que desarrolló el lenguaje de programación Java a principio de los años 90´s y liberó la primera versión en mayo de 1995

tanto una computadora (o dispositivo electrónico) debe tenerla instalada para poder ejecutarlos. La JVM no es un hardware sino un software que se instala en la estación de trabajo o donde se quiere ejecutar una aplicación implementada en este lenguaje (Rojas, 2003).

Jorge Naula plantea que, un marco de trabajo (en inglés framework), es una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación (Gutiérrez, 2006).

Un marco de trabajo, en el desarrollo de software, es una estructura de soporte definida mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros softwares para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio (Ramos Fernández, 2009).

Entre los objetivos principales de un marco de trabajo se encuentran: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo, como el uso de patrones. Los marcos de trabajo utilizados para el desarrollo de la solución son:

- **Swing**: Es el marco de trabajo Java que permite crear aplicaciones de escritorio a partir de componentes comúnmente utilizados en ellas (Java, 2015). Es un conjunto de clases de Java que simplifica la construcción de aplicaciones de escritorio. Permite tener un sistema de ventanas y componentes gráficos, independiente del sistema operativo y la biblioteca de dibujo que se tenga disponible en las estaciones de trabajo clientes. En esta investigación se utiliza para el diseño y construcción de los formularios de la aplicación, debido a las ventajas que ofrece para el desarrollo de componentes que permiten la visualización y registro de los datos.
- **JPA**: Es la Interfaz de Programación de Aplicaciones (API) de persistencia desarrollada para la plataforma Java Enterprise Edition. Es un nuevo estándar de persistencia Java basado en el mapeo objeto-relacional y la utilización de características orientadas a objetos de Java. Brinda un modelo de persistencia basado en objetos comunes conocidos como POJO´s (Plain Old Java Object) para mapear las bases de datos en Java. El mapeo o relación entre entidades Java y tablas de la base de datos se realiza mediante anotaciones en las propias clases de entidad (Yang, 2010).

1.4.5. Entorno de desarrollo integrado

Un Entorno de Desarrollo Integrado, traducido del inglés Integrated Development Environment (IDE) es un programa informático compuesto por un conjunto de herramientas de programación. Puede denominarse como un entorno de programación que ha sido tratado como un programa aplicación y que permite a los programadores escribir, compilar, depurar y ejecutar programas (Castro, 2009). Existen diversos IDEs libres para Java entre los que destacan Eclipse, y Netbeans. Teniendo en cuenta la experiencia del equipo de desarrollo y que no existen diferencias significativas entre ellos, se decide utilizar para el desarrollo de la propuesta de solución Netbeans en su versión 8.1.

Netbeans 8.1: Es un entorno de desarrollo gratuito y de código abierto, que permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones Web, o para dispositivos móviles. Da soporte entre otras a la tecnología Java. Además, puede instalarse en varios sistemas operativos: Windows, Linux, Mac OS, y otros.

Netbeans es también una plataforma de ejecución de aplicaciones, es decir, facilita la escritura de aplicaciones Java, proporcionando una serie de servicios comunes, que a su vez están disponibles a través del IDE. Simplifica la gestión de los proyectos con el uso de diferentes vistas, asistentes de ayuda y estructurando la visualización de manera ordenada, lo que ayuda en el trabajo diario. Sus principales características son (Java, 2015):

- Contiene editores, analizadores de código, y convertidores, los cuales hacen que se puedan actualizar las aplicaciones a utilizar.
- Los datos pueden ser visualizados desde diferentes vistas, pues permite la visión de múltiples ventanas de diferentes proyectos que hayan sido cargados y abiertos en la aplicación.
- Proporciona editores y herramientas integrales para los marcos de trabajo y las tecnologías relacionadas.

Netbeans cuenta con el framework jUnit para realizar pruebas sobre código Java. Es un marco simple para escribir pruebas repetibles. Este conjunto de bibliotecas fue creado por Erich Gamma y Kent Beck. Este tipo de herramientas hacen que el esfuerzo y el trabajo en la fase de pruebas se reduzcan, permitiendo que el desarrollador se centre en la verificación de resultados correctos y no escribiendo código extenso para realizar sus pruebas (Junit, 2013) jUnit está integrado al propio IDE de desarrollo y está diseñado especialmente para Java que es lenguaje de programación elegido para el desarrollo del software.

1.5. Conclusiones del capítulo

La exposición de conceptos relacionados a los softwares educativos y las relaciones de recurrencias, permitió una mejor comprensión de las bases teóricas relacionadas con su definición, características, clasificación y algoritmos de solución. El análisis realizado de los asistentes matemáticos actualmente utilizados en la universidad, permitió corroborar la necesidad de un nuevo sistema para dar solución al problema planteado, que incluya algunas de sus funcionalidades.

Considerando que el equipo de desarrollo es pequeño, que se mantiene una estrecha comunicación con el cliente, que se cuenta con poco tiempo para entregar la propuesta de solución, la experiencia del equipo de desarrollo y que el sistema que se desea obtener es un software portable de pequeñas dimensiones, se seleccionaron como metodología de desarrollo XP y como lenguaje de programación Java. Se escogieron el paradigma de programación y herramientas a utilizar, quedando plasmado un análisis detallado de las mismas para lograr una mejor comprensión de cada una de ellas.

CAPÍTULO 2: PLANIFICACIÓN Y DISEÑO DEL SISTEMA PARA EL CÁLCULO DE RELACIONES DE RECURRENCIA EN LA ASIGNATURA DE MATEMÁTICA DISCRETA II

Este capítulo tiene como objetivo plantear el problema que dará origen al sistema que se va a desarrollar. Se nombran los procesos que se necesitan automatizar. Se definen los requisitos funcionales y no funcionales que el sistema debe respetar y cumplir. Se describen las historias de usuario que permitirán planificar las iteraciones en que se desarrollará el sistema, además del plan de entrega de versiones especificándose la fecha en las que se entregan las versiones del sistema al cliente. Se documentan además, los artefactos correspondientes a la fase de diseño.

2.1. Descripción de las fases de la metodología

La metodología de desarrollo eXtreme Programming comienza con su fase de planificación. Durante esta etapa se realiza el proceso de identificación de los requisitos ya sean funcionales o no funcionales y el desarrollo de las historias de usuario, así como la familiarización del equipo de trabajo con las tecnologías y herramientas seleccionadas para el desarrollo del software. También se acuerda el orden en que deben implementar las historias de usuario y asociadas a éstas, las entregas. Típicamente esta fase consiste en una o varias reuniones grupales de planificación. El resultado de la fase es un plan de entregas donde se realiza una estimación de las versiones que tendrá el producto en su realización, de manera tal que guíe el desarrollo del mismo.

A continuación, le sigue la fase de diseño donde se diseñan aquellas historias de usuario que el cliente ha seleccionado para la iteración actual por dos motivos: por un lado, se considera que no es posible tener un diseño completo del sistema y sin errores desde el principio, el segundo motivo es que, dada la naturaleza cambiante del proyecto, el hacer un diseño muy extenso en las fases iniciales del proyecto para luego modificarlo, se considera un desperdicio de tiempo. El resultado de esta fase es la generación de Tarjetas de Clases- Responsabilidad-Colaboración (CRC), que tienen como principal objetivo ayudar a dejar el pensamiento procedimental para incorporarse al enfoque orientado a objetos. Cada tarjeta representa una

clase con su nombre en la parte superior, en la sección inferior izquierda están descritas las responsabilidades y a la derecha las clases que le sirven de soporte.

La fase de codificación es un proceso que se realiza de forma paralela con el diseño y la cual está sujeta a varias observaciones por parte de XP consideradas controversiales tales como la rotación de los programadores o la programación en parejas. Esta fase genera Cliente siempre presente y Desarrollo en pareja como artefactos (Echeverry Tobón, 2007).

La última fase es la de Prueba, puesto que XP enfatiza mucho los aspectos relacionados con las pruebas, clasificándolas en diferentes tipos y funcionalidades específicas, indicando quién, cuándo y cómo deben ser implementadas y ejecutadas. Del buen uso de las pruebas depende el éxito de otras prácticas, tales como la propiedad colectiva del código y la refactorización. Cuando se tienen bien implementadas las pruebas no habrá temor de modificar el código del otro programador en el sentido que, si se daña alguna sección, las pruebas mostrarán el error y permitirán encontrarlo. El mismo criterio se aplica a la refactorización. Uno de los elementos que podría obstaculizar que un programador cambie una sección de código funcional es precisamente hacer que este deje de funcionar. Si se tiene un grupo de pruebas que garantice su buen funcionamiento, este temor se mitiga en gran medida. Según XP se debe ser muy estricto con las pruebas. Sólo se deberá liberar una nueva versión si esta ha pasado con el cien por ciento de las pruebas. En caso contrario se empleará el resultado de estas para identificar el error y solucionarlo con mecanismos ya definidos, generando pruebas unitarias y pruebas de aceptación como artefactos (Echeverry Tobón, 2007).

2.2. Descripción del dominio

La Matemática Discreta II, es una asignatura que se imparte en la FICI a los estudiantes de primer año de la carrera. En ella se utilizan un conjunto de asistentes matemáticos que permiten complementar el aprendizaje de los contenidos impartidos en dicha materia. Estos asistentes tienen como limitaciones que resultan de difícil manejo por los estudiantes de nuevo ingreso y la notación que utilizan no se ajusta a la establecida en las indicaciones metodológicas de la asignatura. Además, el tratamiento a las relaciones de recurrencia no favorece el proceso de enseñanza- aprendizaje, ya que no permiten la visualización del algoritmo de solución paso a paso, lo que dificulta su comprensión.

Para darle solución a esta situación se implementará una herramienta con el objetivo de contribuir como apoyo al aprendizaje del tema de Relaciones de Recurrencia de Matemática Discreta II. Dicha herramienta permitirá el cálculo de las relaciones de recurrencia lineales homogéneas y lineales no homogéneas, a partir de la implementación de los algoritmos que le dan solución. Para ello deberá iniciar con la clasificación de las relaciones introducidas por el usuario a partir de su linealidad, homogeneidad, orden y naturaleza de los coeficientes, y luego las resolverá brindando las respuestas en un lenguaje entendible por los estudiantes y con la notación establecida por la asignatura. Se tiene como propuesta de solución una aplicación de escritorio con la estructura que se muestra en la Figura 1.

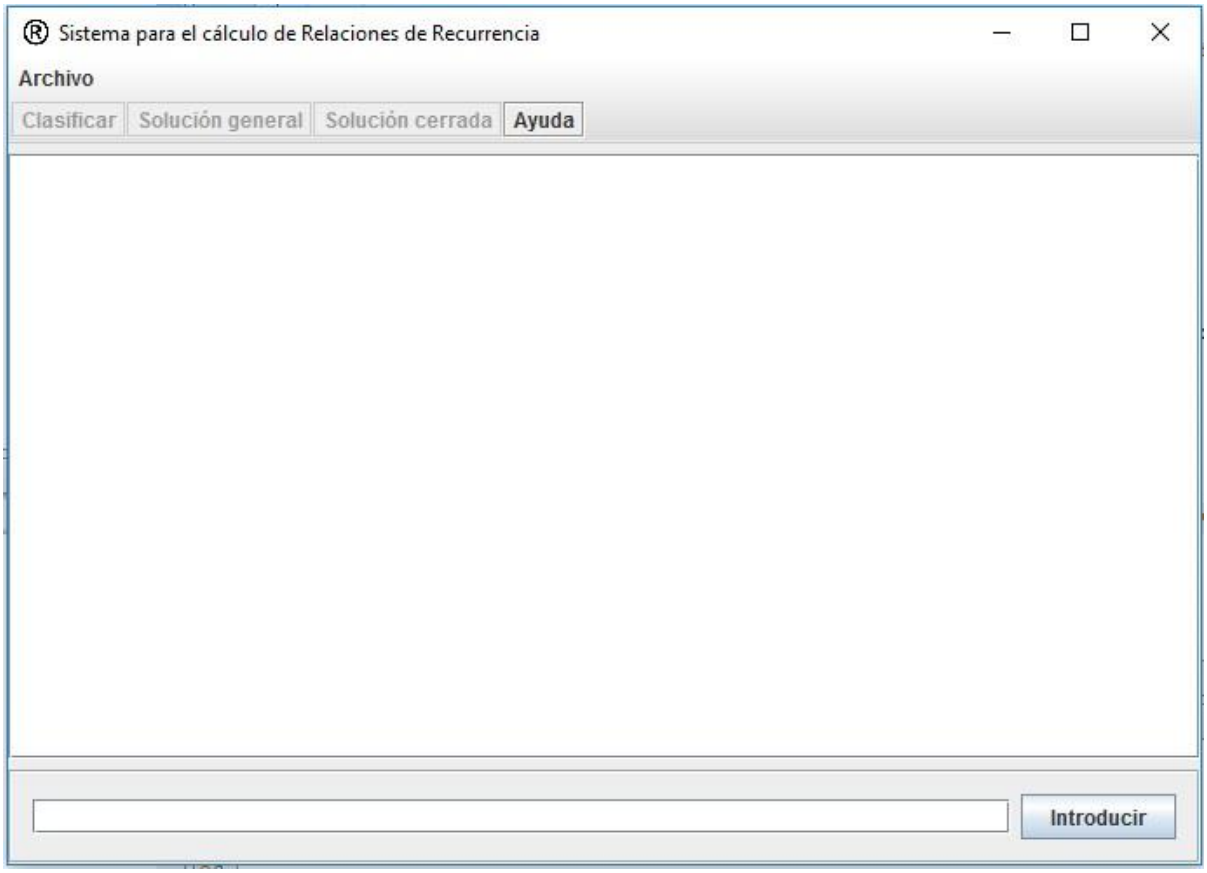


Figura 1. Propuesta de solución (elaboración propia)

2.3. Especificación de los requisitos del software.

Los requisitos para un sistema de software determinan lo que hará el sistema y definen las restricciones de su operación e implementación (Esteller, 2012). Los mismos se clasifican en funcionales y no funcionales.

2.3.1. Requisitos funcionales

Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, la forma en que debe reaccionar ante ciertas entradas y cómo se debe comportar en situaciones particulares (Sommerville, 2005). Se identificaron los siguientes requisitos funcionales:

Requisitos Funcionales

- **RF1.** Guardar datos en fichero.
- **RF2.** Cargar datos en fichero.
- **RF3.** Introducir relación de recurrencia.
- **RF4.** Introducir valores iniciales.
- **RF5.** Clasificar las relaciones de recurrencia según la linealidad.
- **RF6.** Clasificar las relaciones de recurrencia según la homogeneidad.
- **RF7.** Clasificar las relaciones de recurrencia según naturaleza de los coeficientes.
- **RF8.** Clasificar las relaciones de recurrencia según el orden.
- **RF9.** Utilización de la notación usual de las relaciones de recurrencia.
- **RF10.** Generar relación de recurrencia según la linealidad.
- **RF11.** Generar relación de recurrencia según la homogeneidad.
- **RF12.** Generar relación de recurrencia según la naturaleza de los coeficientes.
- **RF13.** Generar relación de recurrencia según el orden.
- **RF14.** Resolver relación de recurrencia.
- **RF15.** Aplicar el método de solución Polinomio Característico.
- **RF16.** Determinar el polinomio y la ecuación.
- **RF17.** Resolver la ecuación característica.
- **RF18.** Obtener las raíces.

- **RF19.** Aplicar el método de solución de los Coeficientes Indeterminados para funciones exponenciales del tipo r^n.
- **RF20.** Obtener la solución general.
- **RF21.** Obtener la solución particular.

2.3.2. Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido, confiable y de fácil uso. Son restricciones de los servicios o funciones ofrecidos por el sistema. Se aplican al sistema en su totalidad y no se refieren directamente a funciones específicas, sino a sus propiedades emergentes; el tiempo de respuesta y la capacidad de almacenamiento son algunas de ellas. Pueden estar dirigidos incluso al proceso de desarrollo, especificando herramientas o estándares de calidad a emplear en función de las necesidades del usuario (Sommerville, 2005). A continuación, se relacionan los requisitos no funcionales identificados para la propuesta de solución:

Requisitos no Funcionales

- **Usabilidad:**

❖ **RnF1.** El sistema debe contar con una interfaz agradable y con botones que tengan nombres sugerentes para que usuarios inexpertos puedan interactuar fácilmente con el software.

❖ **RnF2.** El sistema debe ser usado por cualquier persona con las siguientes características:

- Conocimientos básicos relativos de la Matemática Discreta II
- Conocimientos básicos relativos al uso de una computadora
- Conocimientos básicos del sistema operativo Windows
- Conocimientos básicos del sistema operativo Linux

❖ **RnF3.** El sistema se distribuirá en lenguaje español.

- **Eficiencia:**

❖ **RnF4.** La aplicación debe mantener un tiempo de respuesta aceptable en cuanto a la construcción de objetos, realización de operaciones y representación de diagramas, el tiempo será aceptable cuando se encuentre en el rango entre uno y dos segundos.

- **Hardware:**

Para explotación del cliente:

❖ **RnF5.** PC con un CPU a 133 MHZ o superior.

❖ **RnF6.** Capacidad de almacenamiento en disco duro de 20 GB o superior.

❖ **RnF7.** 1GB de RAM recomendada o superior.

- **Software:**

❖ **RnF8**. La PC donde se instalará la aplicación debe contar con un SO Windows o GNU/Linux.

❖ **RnF9.** La PC donde se instalará la aplicación debe tener instalada la máquina virtual de Java (JVM) en su versión 6.0.

- **Apariencia:**

❖ **RnF10.** Las interfaces de la aplicación contarán con los componentes visuales necesarios para las operaciones correspondientes.

2.3.3. Especificación de requisitos funcionales

El ciclo de vida de un proyecto realizado con la metodología XP se inicia con la etapa de Exploración. En esta etapa, los clientes plantean a grandes rasgos las historias de usuarios. Las historias de usuarios (HU) es el artefacto que utiliza la metodología XP para especificar los requisitos del software y constituyen la base para las pruebas funcionales, son equivalentes a los casos de uso en el proceso unificado. Básicamente una historia de usuario es una lista priorizada de requisitos o funcionalidades, descritas usando la terminología del cliente. Al finalizar el equipo cuenta con suficiente material de trabajo como para producir una primera entrega. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas y tecnologías que serán utilizadas en el proyecto (Letelier, 2006).

En esta etapa el cliente indica las características que debe tener el programa. Tarda pocas semanas y depende de la medida en la que los programadores se adapten con la tecnología.

La especificación de requisitos establece la base para el acuerdo entre clientes y desarrolladores de software, quedando definido el comportamiento deseado del producto (IEEE, 2004).

En la metodología XP las HU son el artefacto que se utilizan para especificar los requisitos funcionales del sistema. A continuación, se describen los campos que se deben llenar en una HU y se muestran las obtenidas (ver Tabla 3 y Anexos del 1 al 17):

- **Número:** identificador de la HU
- **Nombre:** nombre que identifica a la HU
- **Usuario:** involucrados en la ejecución de la HU
- **Iteración asignada:** iteración en que se implementará la HU
- **Prioridad en el negocio:** grado de prioridad que le asigna el cliente a la HU en dependencia del valor en el negocio. Los valores que puede tomar son alta, media o baja
- **Riesgo en desarrollo:** grado de complejidad que le asigna el equipo de desarrollo a la HU luego de analizarla (alto, medio o bajo)
- **Descripción:** descripción sintetizada de la HU

Tabla 3. Descripción de la HU "Introducir relación de recurrencia" (elaboración propia)

Historia de usuario	
Número:1	**Nombre de historia de usuario**: Introducir relación de recurrencia
Modificación de historia de usuario	
Usuario: Usuario	**Iteración asignada**:1
Prioridad en negocio: Alta	
Riesgo en desarrollo: Bajo	
Descripción: Se brinda la posibilidad de que toda persona que acceda al sistema introduzca los datos necesarios para añadir una relación de recurrencia. En caso de que los datos sean correctos se muestra la relación de recurrencia. En caso contrario se le muestra un mensaje de aviso indicándole que los datos introducidos son incorrectos.	

.

2.4. Planificación

En la fase de planificación el cliente establece la prioridad que tendrá cada historia de usuario según sus necesidades. Posteriormente los programadores realizan una estimación del esfuerzo que se necesita para cada una de las historias de usuario, además de tomar acuerdos sobre el contenido de las entregas (Letelier, 2006).

Las estimaciones de esfuerzo asociado a la implementación de las historias de usuario se establecen por parte de los programadores, usando puntos. Un punto, equivale a una semana ideal de programación, lo que se traduce a cinco días de programación, dígase de lunes a viernes, un total de 40 horas de programación. Para el logro del sistema se ha realizado la estimación de esfuerzo por cada historia de usuario (ver Tabla 4).

Tabla 4. Estimación de esfuerzo por HU (elaboración propia)

Historia de usuario	Estimación
Introducir relación de recurrencia.	0.6
Introducir valores iniciales.	0.4
Clasificar las relaciones de recurrencia según su linealidad.	0.2
Clasificar las relaciones de recurrencia según su homogeneidad.	0.2
Clasificar las relaciones de recurrencia según la naturaleza de los coeficientes.	0.2
Clasificar las relaciones de recurrencia según el orden.	0.2
Generar relación de recurrencia según su linealidad.	0.2
Generar relación de recurrencia según su homogeneidad.	0.2
Generar relación de recurrencia según la naturaleza de los coeficientes.	0.2
Generar relación de recurrencia según su orden.	0.2
Resolver relación de recurrencia.	1
Aplicar el método de solución Polinomio Característico.	0.6
Aplicar el método de solución de los Coeficientes Indeterminados para funciones exponenciales del tipo r^n.	0.6
Determinar el polinomio y la ecuación.	0.6
Resolver el polinomio.	0.6

Obtener las raíces.	1
Obtener la solución general.	1
Obtener la solución particular.	1

2.4.1. Plan de iteraciones

El plan de iteraciones muestra cuáles son las historias de usuarios que serán implementadas en cada iteración del sistema y las posibles fechas de liberaciones. En la primera iteración se puede hacer el intento de establecer una arquitectura para el sistema de modo que pueda ser utilizada durante el tiempo total de construcción. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide cuáles son las historias que se implementarán en cada iteración. Al final de la última iteración el sistema estará listo para entrar en producción, tratando de esta manera de tener preparadas las funcionalidades básicas e indispensables del sistema. El sistema que se pretende construir se desarrollará en tres iteraciones, explicadas detalladamente a continuación:

- **Iteración 1:** En esta iteración se implementan las historias de usuarios 1, 2, 3, 4, 5, 6 que se consideraron más necesarias atendiendo a su relevancia e impacto para el cliente. De esta forma se brinda la posibilidad de introducir una relación de recurrencia y clasificarla según sus criterios (linealidad, homogeneidad, naturaleza de los coeficientes y el orden).
- **Iteración 2:** En esta iteración se implementan las historias de usuario con prioridad alta. Brinda la posibilidad de que toda persona que acceda al sistema pueda generar y resolver relaciones de recurrencia. Al finalizar se tendrán las historias de usuario: 7, 8, 9, 10,11, 12, 13, 14, 15, 16.
- **Iteración 3:** En esta iteración se implementan las historias de usuario con menos prioridad. Al finalizar se tendrán implementadas las historias de usuario: 17,18. A partir de este momento quedará conformado totalmente el sistema y será sometido a pruebas.

2.4.2. Plan de duración de iteraciones

Como parte del ciclo de vida de un proyecto utilizando la metodología XP se crea el plan de duración de iteraciones. Para la confección del plan se tiene en cuenta la estimación de esfuerzo para cada HU. Este plan permite mostrar la duración de cada iteración, así como el orden en que se implementarán las historias de usuario, teniendo una mayor organización. Para el plan de duración de iteraciones una semana tiene el equivalente de siete días normales (ver Tabla 5).

Tabla 5. Plan de duración de iteraciones (elaboración propia)

Iteraciones	Orden de las historias de usuario a implementar	Duración total de la iteración
1	1. Introducir una relación de recurrencia. 2. Introducir valores iniciales. 3. Clasificar las relaciones de recurrencia según la linealidad. 4. Clasificar las relaciones de recurrencia según la homogeneidad. 5. Clasificar las relaciones de recurrencia según la naturaleza de los coeficientes. 6. Clasificar las relaciones de recurrencia según el orden.	4 semanas
2	1. Generar relación de recurrencia según su linealidad. 2. Generar relación de recurrencia según su homogeneidad. 3. Generar relación de recurrencia según la naturaleza de los coeficientes. 4. Generar relación de recurrencia según su orden. 5. Resolver relación de recurrencia. 6. Aplicar el método de solución Polinomio Característico. 7. Aplicar el método de solución Coeficientes Indeterminados para funciones exponenciales del tipo r^n. 8. Determinar el polinomio y la ecuación. 9. Resolver el polinomio.	6 semanas

	10. Obtener las raíces.	
3	1. Obtener la solución general. 2. Obtener la solución particular.	2 semanas

2.4.3 Plan de entregas

Al comenzar el proyecto se realiza una reunión entre el equipo de trabajo y los clientes. En dicha reunión se define el marco temporal de la realización del sistema. El cliente expone las historias de usuario a los integrantes del grupo, quienes estimarán el grado de dificultad de la implementación de cada historia. Las historias de usuario son asignadas a las diferentes iteraciones según su orden de relevancia para el proyecto. En el proceso de selección de las historias de usuario para cada iteración, se tiene en cuenta que la suma de las estimaciones sea aproximada a la velocidad del proyecto de la iteración pasada. En esta reunión se predicen los tiempos que se utilizarán en la realización de las diferentes etapas del proyecto, los cuales no son datos exactos, pero proporcionan una base del cronograma. Con el plan de entrega se da un aproximado de las entregas de las versiones teniendo en cuenta que la implementación comienza a finales de abril (ver Tabla 6).

Tabla 6. Plan de entregas (elaboración propia)

A entregar	Final 1ra iteración	Final 2da iteración	Final 3ra iteración
Herramienta de apoyo a la enseñanza de las Relaciones de Recurrencia en la asignatura Matemática Discreta II	1/03/2016	2/04/2016	30/05/2016

2.5. Arquitectura de la solución

“La arquitectura de un sistema es un marco conceptual completo que describe su forma y estructura (sus componentes y la manera en que se integran) (Pressman, 2005). Para lograr la calidad del software es necesario establecer desde el inicio una arquitectura que garantice robustez y escalabilidad.

La arquitectura de software está formada por la estructura de los elementos de un programa o sistema, sus interrelaciones y los principios y reglas que gobierna su diseño y evolución a lo largo del tiempo. También es definida por David Garlan y Mary Shaw como un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema". Para desarrollar una arquitectura potente se utilizan estilos arquitectónicos, éstos indican los tipos de componentes y conectores involucrados, así como patrones y restricciones de interconexión o composición entre ellos (Oktaba, 2010).

Estilo N Capas

El estilo N Capas está basado en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades las funcionalidades que implementan. N capas es un estilo arquitectónico donde cada capa puede ser modificada tanto como sea necesario sin provocar cambios en las demás. Esto permite el desarrollo de aplicaciones más robustas debido al encapsulamiento. Es factible un mantenimiento y soporte más sencillo, así como una mayor flexibilidad (De la Torre Llorent, 2003).

La arquitectura que se definió para el desarrollo del sistema estará basada en tres capas, las cuales son: Interfaz de Usuario (IU), Dominio y Negocio como se muestra en la Figura 2.

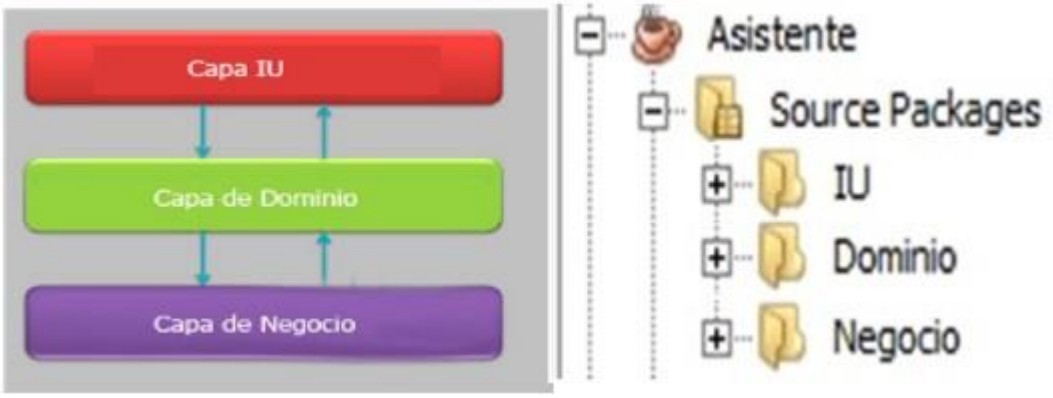


Figura 2. Arquitectura N capas (elaboración propia)

- La capa de Interfaz de Usuario (IU) contiene los formularios con los que interactúan los usuarios del sistema y permite la visualización de toda la información que estos necesitan y las acciones que gestionan las funcionalidades

de los formularios. Básicamente, se responsabiliza de que se le comunique información al usuario por parte del sistema y viceversa, manteniendo una comunicación exclusiva con la capa de Dominio.

- La capa de Dominio es la capa que contiene las funcionalidades a realizar con la información recibida desde la capa de IU, responsabilizándose además de que se le envíen las respuestas adecuadas a dicha capa. Contiene la mayor parte de la implementación de la lógica de negocio, responsabilizándose de las tareas propias de la aplicación, controlando las reglas de negocio y las entradas del usuario. Es la capa encargada de procesar los datos matemáticos.
- La capa de Negocio contiene las clases persistentes para la gestión de los datos. Es responsable de la gestión y almacenamiento permanente de los datos.

2.6. Diseño

XP establece prácticas sobre el diseño para lograr un sistema robusto y reutilizable manteniendo su simplicidad, logrando que cambios o modificaciones futuras puedan realizarse de manera más sencilla. XP no define una técnica específica de modelado, pueden utilizarse indistintamente esquemas sencillos, tarjetas CRC o diagramas de clase utilizando UML, siempre que sean útiles y no requieran mucho tiempo en su creación (Beck, 1999).

Los cuatro valores de la metodología XP: comunicación, simplicidad, retroalimentación y coraje, inciden directamente sobre la estrategia de diseño. Se debe crear un diseño simple donde cada elemento existente represente aspectos importantes del software a implementar, fácil de comunicar y confiar en que solo con lo diseñado es suficiente en ese mismo instante; que en el momento que se desee y sea necesario se puede incrementar el diseño.

2.6.1. Patrones de diseño

Los patrones de diseño son descripciones de clases y objetos relacionados que están particularizados para resolver un problema de diseño general en determinado contexto (Gamma Erich, 2003).

Un patrón de diseño nomina, abstrae e identifica los aspectos clave de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. El

patrón de diseño identifica las clases e instancias participantes, sus roles y colaboraciones y la distribución de responsabilidades (Gamma Erich, 2003).

Patrones GRASP: Son patrones generales de software para la asignación de responsabilidades a objetos. Describen los principios fundamentales del diseño de objetos para la asignación de responsabilidades (Larman, 2004).

- **Experto**: Este patrón indica que la implementación de un determinado método o la creación de un objeto debe recaer sobre la clase que conoce toda la información para crearlo (Larman, 2004). En la aplicación se evidencia este patrón en la clase Término (ver Figura 3).

```
public class Termino implements Comparable<Termino> {

    private Token signo;
    private LinkedList<Token> elementos;

    public Termino(Token signo, LinkedList<Token> elementos) {
        this.signo = signo;
        this.elementos = elementos;

    }
Termino termino= new Termino(signo, elementos);

    public Token getSigno() {
        return signo;
    }
```

Figura 3. Clase donde se evidencia el patrón experto (elaboración propia)

- **Alta Cohesión**: Este patrón es utilizado para evita asignar demasiadas responsabilidades a las clases. Brinda la posibilidad de que la información que se almacene en cada clase sea coherente (Larman, 2004).
- **Bajo Acoplamiento**: Este patrón es el encargado de disminuir la dependencia entre clases, posibilitando que la modificación en alguna de ellas repercuta lo menos posible en las demás clases (Larman, 2004).

2.6.2. Tarjetas CRC (Clase-Responsabilidad-Colaboración)

La metodología XP no obliga a la realización de diagramas UML para representar las clases que contendrán la lógica del negocio. En su lugar propone el uso de tarjetas CRC como técnica de modelado para ayudar a los desarrolladores de software a crear diseños de clases orientados a responsabilidades.

Las tarjetas CRC son fichas, una por cada clase, en las que se escriben brevemente las responsabilidades de la clase y una lista de los objetos con los que colabora para llevar a cabo esas responsabilidades (Larman, 2005).

A continuación, se muestra una tarjeta CRC correspondientes al sistema (ver Tabla 7 y Anexos del 18 al 41):

Tabla 7. Tarjeta CRC clase RelacionRecurrencia (elaboración propia)

RelacionRecurrencia	
Responsabilidad	**Colaboraciones**
Clase que se encarga de brindar los elementos que conforman una relación de recurrencia. Contiene y manipula los siguientes atributos que hacen persistir los datos en memoria, mientras esta se encuentra en ejecución: **Termino términos**; **List**<**Termino**> terminos;	

2.6.3. Prototipos de interfaz de usuario

Un prototipo es una representación o demostración de un sistema. Los prototipos reducen el riesgo de construir productos que no satisfagan las necesidades de los clientes ya que si al usuario no le gusta una parte del prototipo este problema se corrige hasta que el cliente quede satisfecho. Los prototipos son un mecanismo para clarificar qué es lo que se quiere o es posible (Larman, 2005).

Principios que se tuvieron en cuenta para el diseño de interfaces de usuario (IU):

1. **Anticipación**: Las aplicaciones deberían intentar anticiparse a las necesidades del usuario y no esperar a que el usuario tenga que buscar la información, recopilarla o invocar las herramientas que va a utilizar (M. Gómez, 2002).

2. **Autonomía:** La computadora, la IU y el entorno de trabajo deben estar a disposición del usuario. Se debe dar al usuario el ambiente flexible para que pueda aprender rápidamente a usar la aplicación. Sin embargo, está comprobado que el entorno de trabajo debe tener ciertas cotas, es decir, ser explorable pero no azaroso (M. Gómez, 2002).

3. **Consistencia:** Para lograr una mayor consistencia en la IU se requiere profundizar en diferentes aspectos que están catalogados en niveles. Se realiza un ordenamiento de mayor a menor consistencia:

1. Interpretación del comportamiento del usuario*:* la IU debe comprender el significado que le atribuye un usuario a cada requerimiento. Ejemplo: mantener el significado de los comandos abreviados definidos por el usuario (M. Gómez, 2002).
2. Estructuras invisibles: se requiere una definición clara de las mismas, ya que sino el usuario nunca podría llegar a descubrir su uso. Ejemplo: la ampliación de ventanas mediante la extensión de sus bordes (M. Gómez, 2002).
3. Pequeñas estructuras visibles: se puede establecer un conjunto de objetos visibles capaces de ser controlados por el usuario, que permitan ahorrar tiempo en la ejecución de tareas específicas. Ejemplo: ícono y/o botón para impresión.
4. Una sola aplicación o servicio: la IU permite visualizar a la aplicación o servicio utilizado como un componente único. Ejemplo: La IU despliega un único menú, pudiendo además acceder al mismo mediante comandos abreviados (M. Gómez, 2002).
5. Un conjunto de aplicaciones o servicios: la IU visualiza a la aplicación o servicio utilizado como un conjunto de componentes. Ejemplo: La IU se presenta como un conjunto de barras de comandos desplegadas en diferentes lugares de la pantalla, pudiendo ser desactivadas en forma independiente (M. Gómez, 2002).
6. Consistencia del ambiente: la IU se mantiene en concordancia con el ambiente de trabajo. Ejemplo: La IU utiliza objetos de control como el menú, botones de comandos de manera análoga a otras IU que se usen en el ambiente de trabajo (M. Gómez, 2002).

7. Consistencia de la plataforma: La IU es concordante con la plataforma. Ejemplo: La IU tiene un esquema basado en ventanas, el cual es acorde al manejo de los sistemas operativos con entornos gráficos (M. Gómez, 2002).

4. **Eficiencia del Usuario**: Se debe considerar la productividad del usuario antes que la productividad de la máquina. Si el usuario debe esperar la respuesta del sistema por un período prolongado, estas pérdidas de tiempo se pueden convertir en pérdidas económicas para la organización. Los mensajes de ayuda deben ser sencillos y proveer respuestas a los problemas. El menú y las etiquetas de botones deberían tener las palabras claves del proceso.

A continuación, se muestran los prototipos de interfaces de usuarios correspondientes a las HU de la primera iteración (ver Figuras 4 y 5):

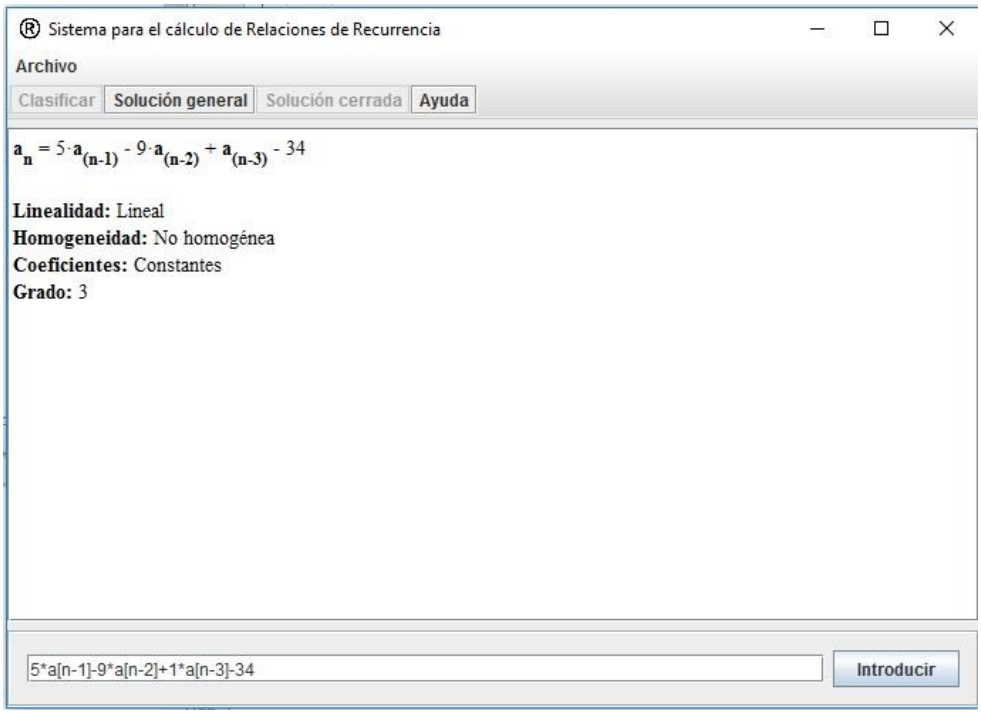


Figura 4. Clasificar relación de recurrencia según sus criterios (elaboración propia)

Figura 5. Introducir valores iniciales (elaboración propia)

A continuación, se muestran dos prototipos de interfaces de usuarios correspondientes a las HU de la tercera iteración (ver Figuras la 6 y 7):

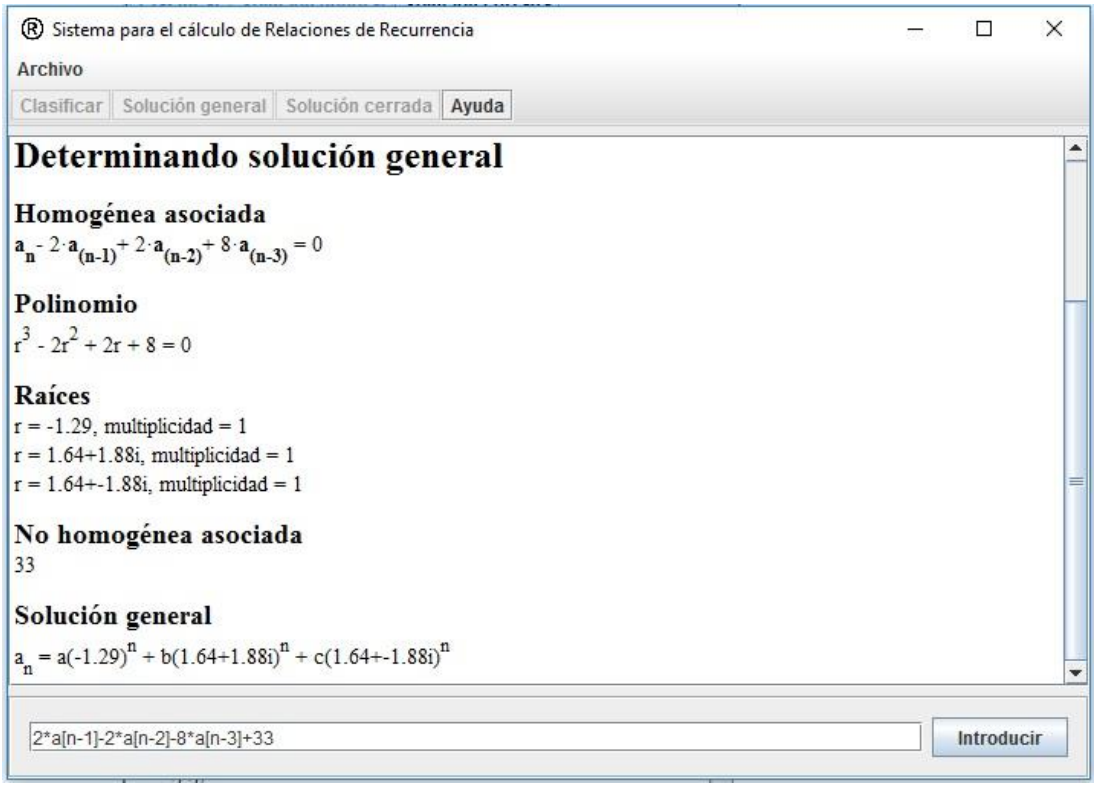


Figura 6. Determinar la solución general de una relación de recurrencia lineal, homogénea de coeficientes constantes (elaboración propia).

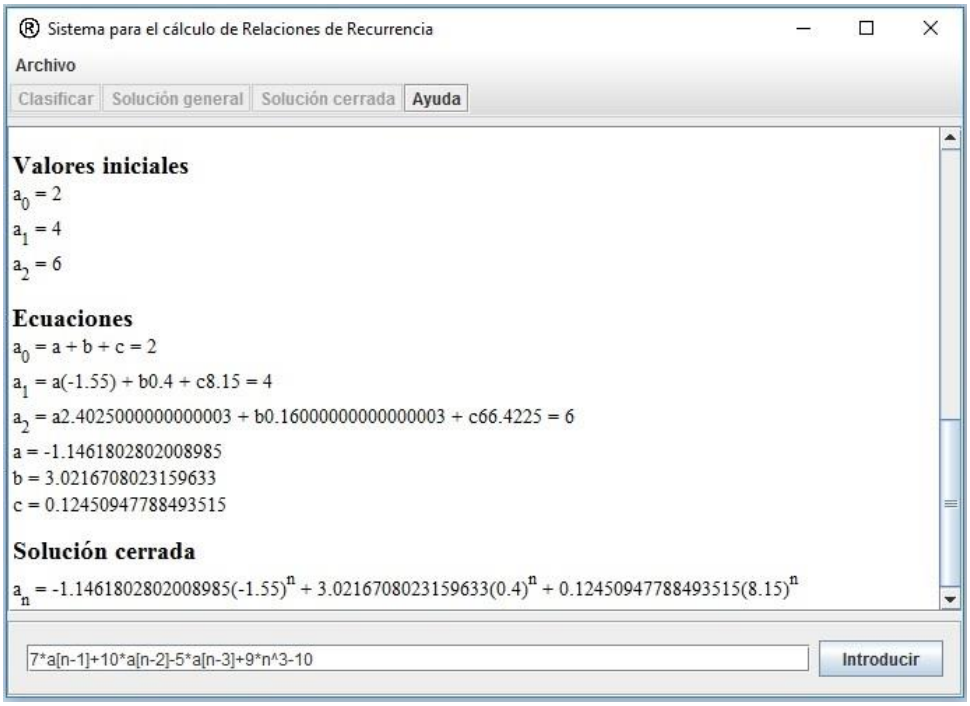


Figura 7. Determinar la solución particular de una relación de recurrencia lineal, homogénea de coeficientes constantes (elaboración propia).

2.7. Conclusiones parciales

Con el presente, quedaron definidos los elementos fundamentales para la construcción del sistema. Como parte de la fase de planificación se elaboró el plan de iteraciones y el plan de entregas, los cuales facilitaron la planificación y organización en el proceso de desarrollo.

Se definieron en total veintiún requisitos funcionales y diez no funcionales para describir las características y requisitos del sistema, lo que contribuyó a la comprensión y visualización de los requisitos del cliente. A partir de la obtención de los requisitos funcionales se modelaron, a través de dieciocho HU, las funcionalidades del sistema.

Se definió como patrón arquitectónico el estilo arquitectónico en capas, pues es el que mejor se ajusta a las características del sistema, ya que se pueden realizar modificaciones en cada una de las capas sin provocar cambios en las restantes capas y se agrupan las clases en paquetes para obtener una mejor organización de las mismas. Por otro lado, se describieron las clases presentes en la solución con el objetivo de facilitar su comprensión, utilizando un total de 24 tarjetas CRC.

CAPÍTULO 3: CODIFICACIÓN Y PRUEBA DEL SISTEMA PARA EL CÁLCULO DE RELACIONES DE RECURRENCIA EN LA ASIGNATURA DE MATEMÁTICA DISCRETA II

En el presente capítulo se documentan las fases de codificación y prueba según lo propone la metodología de desarrollo XP. Se desglosan las HU en tareas con el objetivo de facilitar el trabajo de los desarrolladores. Se muestran los resultados de las pruebas de aceptación realizadas a la aplicación. Se realiza la verificación y validación de la propuesta de solución.

3.1. Codificación

En esta fase se implementan las historias de usuario, las que se fragmentan en tareas de programación para ayudar a organizar la implementación del sistema. Estas tareas son descritas y utilizadas por los programadores como guía para el desarrollo del sistema. En esta etapa XP propone tener en cuenta una serie de aspectos como son la disponibilidad del cliente y el desarrollo en pareja para lograr mayores resultados en la implementación del software (Beck, 1999).

- **Disponibilidad del cliente:** Uno de los requerimientos de XP es que el cliente esté siempre disponible, no solamente para solucionar las dudas del grupo de desarrollo, sino que debería ser parte de éste. En este sentido se convierte en gran ayuda al solucionar todas las dudas que puedan surgir, especialmente cara a cara, para garantizar que lo implementado cubre con las necesidades planteadas en las historias de usuario (Echeverry Tobón, 2007). El cliente formó parte del equipo de desarrollo, describió las HU, guio la toma de decisiones, aprobó las versiones del producto y verificó el cumplimiento de los objetivos trazados.
- **Desarrollo en pareja:** Todo el código debe ser creado por parejas de programadores sentados ambos frente a un único computador lo que en principio representa una reducción de un 50% en productividad, sin embargo, según XP no es tal la pérdida. Se entiende que no hay mucha diferencia, en lo que a la cantidad se refiere, entre el código producido por una pareja bajo estas condiciones que el creado por los mismos miembros trabajando en forma separada, con la excepción que uno o ambos programadores sean muy expertos

en la herramienta en cuestión. Cuando se trabaja en parejas se obtiene un diseño de mejor calidad y un código más organizado y con menores errores que si se trabajase solo, además de la ventaja que representa contar con un compañero que ayude a solucionar inconvenientes en tiempo de codificación, los cuales se presentan con mucha frecuencia. Se recomienda que mientras un miembro de la pareja se preocupa del método que se está escribiendo, el otro se ocupe de cómo encaja éste en el resto de la clase (Echeverry Tobón, 2007). Toda la implementación fue realizada por dos personas que trabajaron en forma conjunta.

3.1.1. Tareas de ingeniería

Las Tareas de ingeniería (TI) también conocidas como tareas de implementación permiten a los desarrolladores obtener un nivel de detalle más avanzado que el que propician las HU. Se usan para describir las tareas que se realizan. Estas tareas tienen relación con una historia de usuario. Se especifica la fecha de inicio y fin de la tarea, se nombra al programador responsable de cumplirla y se describe lo que se tratará de hacer en la tarea (Pinciroli, 2007).

El desarrollo del software se planificó en tres iteraciones de trabajo. Para la implementación de las HU correspondientes a la primera iteración se definieron las siguientes TI (ver Tabla 8):

Iteración 1 - HU # 1

Tabla 8. Tarea de ingeniería: Introducir una relación de recurrencia (elaboración propia)

Tarea de ingeniería	
Número de tarea:1	**Número de la HU:**1
Nombre de tarea: Introducir una relación de recurrencia.	
Tipo de tarea: Desarrollo	
Fecha inicio: 01/03/2016	**Fecha fin:**03/03/2016
Programador responsable: Yoel Pimienta Rivas	
Descripción El sistema muestra la opción para que el usuario pueda introducir una relación de recurrencia.	

Las HU definidas para el resto de las TI de la primera iteración y de las iteraciones 2 y 3 fueron divididas en las siguientes TI (ver Anexos del 42 al 58)

3.2. Pruebas realizadas al sistema

Una vez concluida la implementación del sistema se desarrollaron una serie de pruebas para comprobar que se cumplieron los requerimientos descritos por el cliente. Uno de los pilares fundamentales de la metodología XP es la fase de Prueba. XP divide esta fase en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores; y pruebas de aceptación o pruebas de funcionalidades destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente (Rodríguez, 2007).

Una prueba de software es la ejecución de programas de software con el objetivo de detectar defectos y fallas. Permite comprobar y mostrar la calidad de un producto de software (Pressman, 2005).

Las pruebas tienen los siguientes objetivos (Pressman, 2005):

1. La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.
2. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
3. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Las pruebas constituyen un elemento primordial en la metodología XP y deben aplicarse de forma temprana y frecuente.

Para la realización de las pruebas se establece una estrategia de pruebas. Para la verificación del sistema se usan las pruebas unitarias para probar el código de la aplicación y para la validación del sistema se usan las pruebas de aceptación para demostrar la conformidad de los requisitos.

3.2.1. Pruebas unitarias

Estas pruebas se aplican a todos los métodos no triviales de todas las clases del proyecto con la condición que no se liberará ninguna clase que no tenga asociada su correspondiente paquete de pruebas. Uno de los elementos más importantes en estas es que idealmente deben ser construidas antes que los métodos mismos, permitiéndole al programador tener máxima claridad sobre lo que va a programar antes de hacerlo, así como conocer cada uno de

los casos de prueba que deberá pasar, lo que optimizará su trabajo y su código será de mejor calidad. Deben ser construidas por los programadores con el empleo de algún mecanismo que permita automatizarlas de modo tal que tanto su implementación y ejecución consuman el menor tiempo posible permitiendo sacarles el mejor provecho. Los empleos de pruebas unitarias completas facilitan la liberación continua de versiones por cuanto al implementar algo nuevo y actualizar la última versión, solo es cuestión de ejecutar de forma automática las pruebas unitarias ya creadas para saber que la nueva versión no contiene errores (Echeverry Tobón, 2007).

En este caso se empleó la técnica del camino básico de modo tal que tanto su implementación como su ejecución consumieran el menor tiempo posible, permitiendo sacarles el mayor provecho.

Con la realización de las pruebas unitarias los desarrolladores examinaron que los métodos implementados dieran solución al propósito con el que se confeccionaron. Para ello se utilizó el framework Junit como mecanismo de automatización del cual se obtuvieron los siguientes resultados (ver Figura 8):

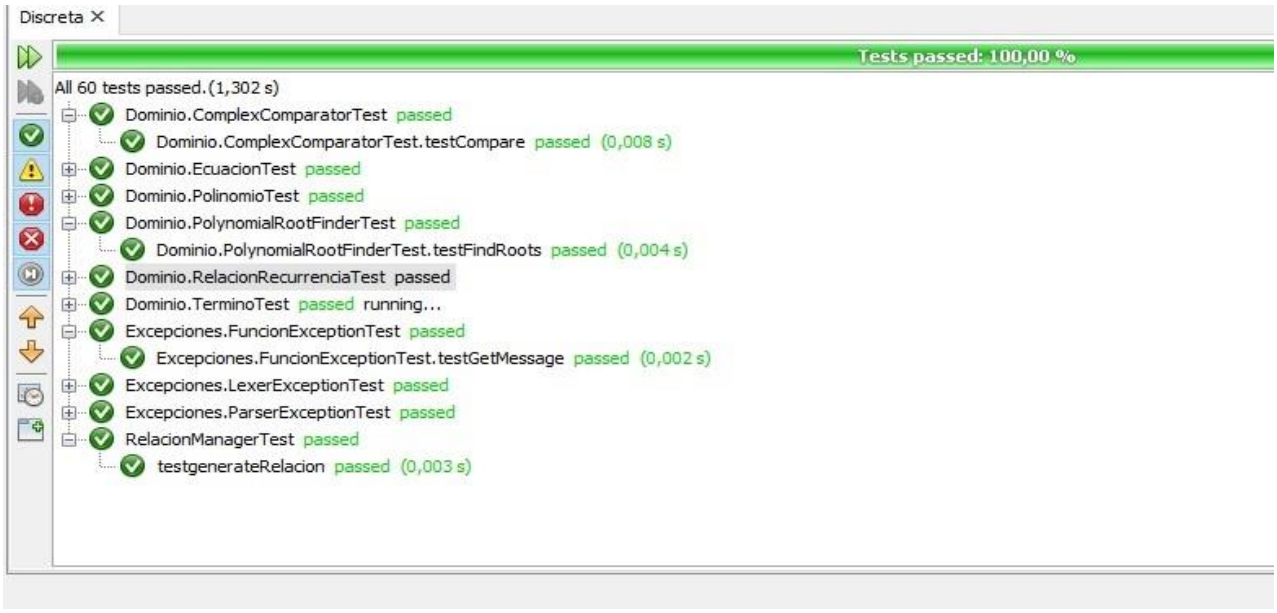


Figura 8. Resultados del mecanismo de pruebas automatizadas aplicadas con el framework Junit (tomado del Junit)

Aplicación de la Técnica Camino Básico:

Las pruebas de caja blanca se aplicaron haciendo uso de la técnica del camino básico. La técnica camino básico es aplicada a todos los métodos implementados. Para este caso, se expone su aplicación sobre la funcionalidad encargada de clasificar las relaciones de recurrencia según la linealidad, el método isLineal. Se identificaron diez bloques de ejecución para este método atendiendo a las dependencias procedimentales, con el objetivo de evaluar la complejidad lógica de un diseño procedimental y usar esta medida como guía para la definición de un conjunto básico de caminos de ejecución.

Esta prueba permite garantizar que en los casos de prueba obtenidos a través del camino básico se ejecute cada sentencia del programa por lo menos una vez (Pressman, 2005). El uso de esta técnica es mostrado en el ejemplo siguiente (ver Figura 9). Se analizan y enumeran las sentencias de código del método isLineal contenido en la clase Término.

```
public boolean isLineal() {
    boolean result = true;
    if (cantAn() > 1) {
        result = false;
    } else {
        for (int i = 1; i < elementos.size() - 1; i++) {
            Token token = elementos.get(i);
            if (token.isAn()) {
                if (elementos.get(i - 1).isPotencia() || elementos.get(i + 1).isPotencia()) {
                    result = false;
                }
            }
            if (token.getKind() == TokenKind.tk_Division) {
                if (elementos.get(i + 1).isAn()) {
                    result = false;
                }
            }
        }
    }
    return result;
}
```

Figura 9. Método isLineal (elaboración propia)

Luego, es representado el grafo de flujo, asociado al código anterior, a través de nodos, aristas y regiones (ver Figura 10):

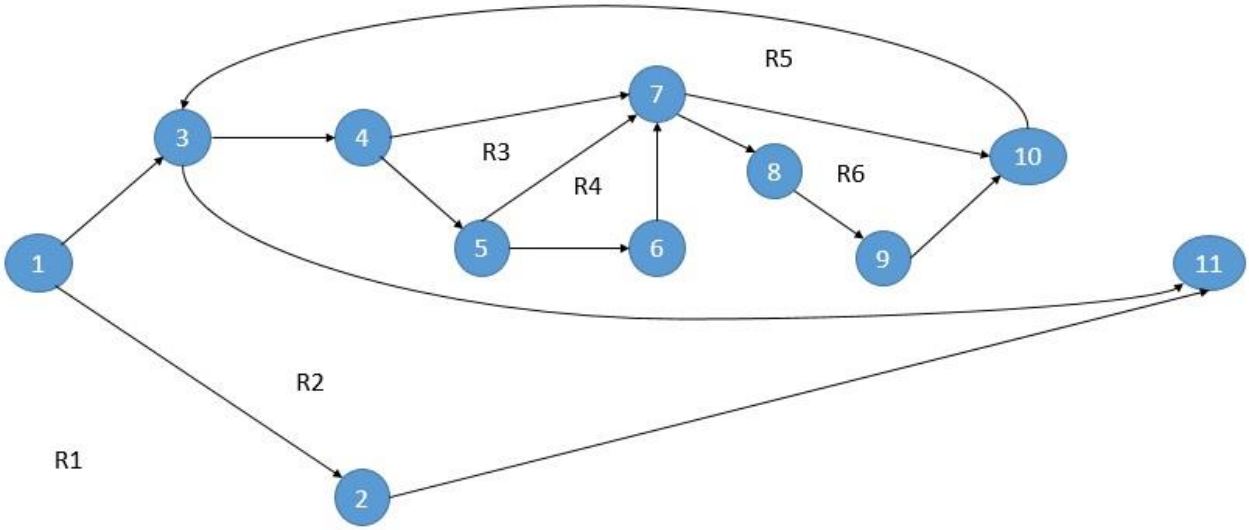


Figura 10. Grafo de flujo del método isLineal (elaboración propia)

Una vez definido el grafo de flujo se procede al cálculo de la complejidad ciclomática siendo esta una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa (Pressman, 2005). Para esta operación existen tres vías de solución, las cuales se enuncian a continuación:

- ✓ **V(G)=(A-N) +2**
- ✓ **V(G)=P +1**
- ✓ **V(G)=R**

La complejidad ciclomática, V (G), de un grafo de flujo G, se define como:

V (G)= A–N+2, donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.

La complejidad ciclomática, V (G), de un grafo de flujo G, también se define como:

V (G)= P + 1, donde P es el número de nodos predicados contenidos en el grafo de flujo G.

V (G)= R, donde R es la cantidad total de regiones.

Del grafo de flujo G se obtiene que:

A = 15	V (G) = A-N+2	P= 5	R=6
N = 11	V (G) = 15-11+2	V (G) = P+1	V (G) = R

V (G) =6 V (G) = 5+1 V (G) = 6

V (G) =6

Se aplican las tres formas para afirmar un resultado seguro y confiable. Luego se obtiene una complejidad ciclomática V (G)=6. Esta cifra representa la cantidad de caminos independientes para el grafo de flujo construido para el método. Luego de tener elaborado el grafo de flujo e identificados los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos (ver Tabla 9). Se escogen los datos de manera que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino. A continuación, se especifican estos casos de prueba.

Camino básico 1: 1, 2,11.

Camino básico 2: 1, 3, 11.

Camino básico 3: 1, 3, 4, 5, 6, 7, 8, 9, 10, 3, 11.

Camino básico 4: 1, 3, 4, 7, 10, 3, 11.

Camino básico 5:1, 3, 4, 7, 8, 9, 10, 3, 11.

Camino básico 6:1, 3, 4, 5, 7, 8, 9, 10, 3, 11.

Tabla 9. Caso de prueba camino básico # 1 (elaboración propia)

Caso de prueba camino básico # 1
Descripción: Este método clasifica cada término de la relación de recurrencia por el criterio de linealidad, la cual puede ser lineal o no lineal.
Condiciones de ejecución: La cantidad de términos debe ser mayor que uno.
Entradas\Pasos de ejecución:
La cantidad de términos debe ser mayor que uno.
Resultado esperado: La relación de recurrencia se clasifica como no lineal.
Evaluación de la prueba: Satisfactoria.

En la siguiente gráfica se muestran los resultados de las pruebas unitarias, a través del método de caja blanca aplicando la técnica de camino básico al código (ver Figura 11).

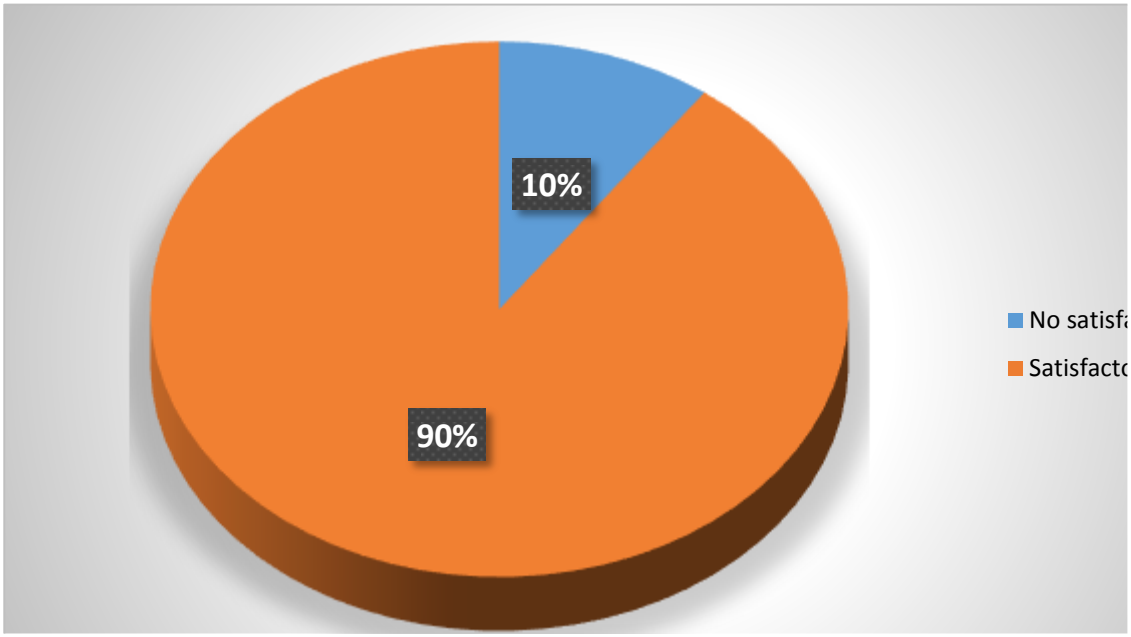


Figura 11. Resultados de las pruebas unitarias (elaboración propia)

Se realizaron 40 casos de prueba de caja blanca (pruebas unitarias), de los cuales 36 resultaron satisfactorios representando el 90% del total de los casos de prueba. Se detectaron un total de cuatro errores, ligados a funcionalidades del sistema, a los que se les dio el tratamiento requerido para el logro de un 100% de pruebas satisfactorias.

En los Anexos del 78 al 82 se muestran los casos de prueba relacionados con el método isLineal, que al ser uno de los más complejos ilustra el procedimiento aplicado en los restantes 34 casos de pruebas realizados.

3.2.2. Pruebas de aceptación

Las pruebas de aceptación resultan de gran importancia, ya que significan la conformidad del cliente con el producto desarrollado, representan el fin de una iteración y el principio de la otra (Rodríguez, 2007).

Las pruebas de aceptación son pruebas de caja negra definidas por el cliente para cada historia de usuario, y tienen como objetivo asegurar que las funcionalidades del sistema cumplen con lo que se espera de ellas. En efecto, las pruebas de aceptación corresponden a

una especie de documento de requerimientos en XP, ya que marcan el camino a seguir en cada iteración, indicándole al equipo de desarrollo hacia donde tiene que ir y en qué puntos o funcionalidades debe poner el mayor esfuerzo y atención (Malfará, 2006).

La prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, o sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene (Pressman, 2006). Las pruebas de aceptación son creadas a partir de las HU. Una HU puede tener tantas pruebas de aceptación como sean necesarias para comprobar su correcto funcionamiento. La HU no se considera completa hasta que no supera todas las pruebas de aceptación.

Estas pruebas deben realizarse con la mayor rapidez posible, para que los desarrolladores puedan realizar los cambios necesarios y el tiempo de desarrollo no se vea afectado gravemente.

Para la realización de las pruebas se describieron los casos de pruebas, uno por cada historia de usuario. Al aplicarse la técnica de partición de equivalencia se determina cada uno de los flujos posibles del sistema y qué resultados debe arrojar en ellos. Por ello cada uno de los casos de prueba evalúa todas las posibles variantes de funcionamiento de las HU (ver Tabla 10).

Tabla 10. Caso de prueba de aceptación HU13_P1 (elaboración propia)

Caso de prueba de aceptación	
Código: HU13_P1	HU: HU13
Nombre: Resolver relación de recurrencia.	
Descripción: Muestra la resolución de la relación de recurrencia.	
Condiciones de ejecución: El usuario puede visualizar la resolución de la relación de recurrencia introducida previamente.	
Entradas\Pasos de ejecución:	
1. El usuario introduce en la caja de texto la relación de recurrencia. 2. El usuario puede o no introducir valores iniciales. 3. Luego se muestra una ventana listando todas las clasificaciones (linealidad, homogeneidad, naturaleza de los coeficientes, y el orden).	

4. Luego de la clasificación, se aplica el método de solución correspondiente.
Resultado esperado: Se muestra la resolución según la relación de recurrencia.
Evaluación de la prueba:

El resto de los casos de pruebas de aceptación se encuentran en los Anexos del 59 al 77.

En la siguiente gráfica se presentan los resultados de las pruebas de aceptación, a través del método de caja negra, aplicadas en tres iteraciones (ver Figura 12):

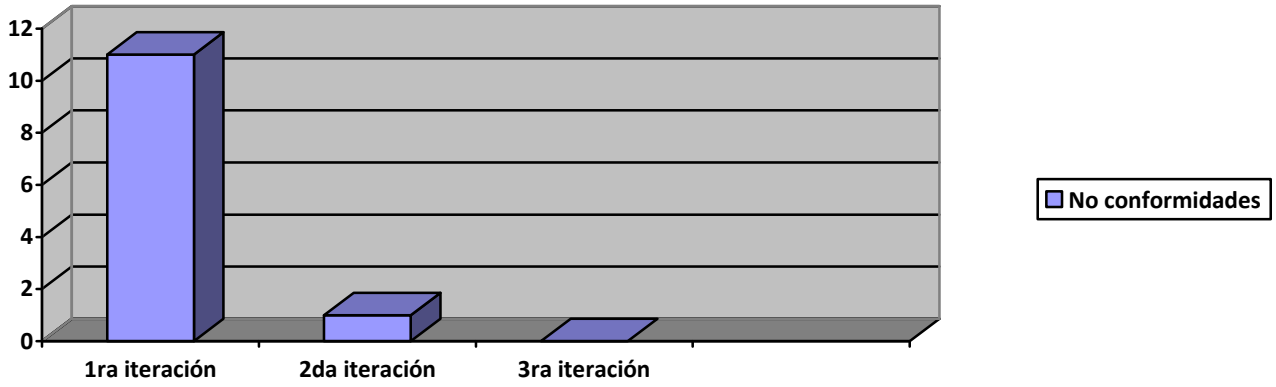


Figura 12. Resultados de las pruebas de aceptación (elaboración propia)

Como se puede observar, en la primera iteración se encontraron un total de once no conformidades, las cuales fueron corregidas satisfactoriamente. En la segunda iteración se detectó una no conformidad que fue corregida. Por último, se realizó una última iteración para validar el correcto funcionamiento del sistema, que tuvo como resultado la no existencia de nuevas no conformidades, emitiéndose por parte del cliente una Carta de aceptación y por el grupo de calidad de CEGEL, el Acta de liberación interna de productos de software como evidencias (ver Anexos 83 y 84).

Las no conformidades encontradas en las diferentes iteraciones se clasifican en ortografía, validación, funcional e interfaz de usuario como se muestra en la Tabla 11.

Tabla 11. Clasificación de las no conformidades (elaboración propia)

Iteración/NC	Interfaz	Ortografía	Validación	Redacción	Funcionalidad
1ra iteración	2	3	2	3	1

2da iteración	0	1	0	0	0
3ra iteración	0	0	0	0	0

De manera general, estas tres iteraciones realizadas en el sistema permitieron la identificación de un total de doce no conformidades, a las cuales se les dio el tratamiento requerido para el logro de un 100% de pruebas satisfactorias.

3.3. Conclusiones parciales

En este capítulo se realizaron las tareas de ingeniería que dieron solución a las HU logrando una mayor organización y rapidez en el desarrollo del sistema. Se realizó la verificación y validación del sistema, analizando los resultados obtenidos en cada prueba realizada al sistema, obteniendo los siguientes resultados:

- A partir de las pruebas unitarias realizadas, se verificó que el sistema mantuviera un correcto funcionamiento. Se detectaron un total de cuatro errores, ligados a funcionalidades del sistema, a los que se les dio el tratamiento requerido para el logro de un 100% de pruebas satisfactorias
- Las pruebas de aceptación validaron que el sistema desarrollado responde y satisface las demandas del cliente, ya que se realizaron tres iteraciones de prueba como se muestra en la Figura 12, las cuales arrojaron un total de 12 no conformidades que fueron corregidas por el equipo de desarrollo en la menor cantidad de tiempo posible, lo que demuestra que se cumplieron los objetivos trazados.

CONCLUSIONES GENERALES

En esta investigación se desarrolló un sistema informático para el cálculo de relaciones de recurrencia en la asignatura de Matemática Discreta II de la carrera de Ingeniería en Ciencias Informáticas. Para ello se han cumplido los aspectos siguientes:

1. Se estableció un marco conceptual de referencia para soportar los fundamentos teóricos de la investigación y dominar cada aspecto de la propuesta, lo que permitió definir y caracterizar los softwares educativos y las relaciones de recurrencia con su clasificación, así como los algoritmos que le dan solución. Se analizaron los asistentes matemáticos actualmente utilizados en la enseñanza de la Matemática Discreta II, que permitieron confirmar la necesidad del desarrollo de esta herramienta.
2. Atendiendo a las características del equipo de desarrollo y de los requerimientos del sistema obtenidos a partir de las reuniones con el cliente y el análisis realizado, se seleccionó como metodología de desarrollo XP. Esta orientó el proceso de desarrollo a través de sus fases de planificación, diseño y codificación, en las que se obtuvieron veintiún requisitos funcionales y diez no funcionales que se expresaron en un total de dieciocho historias de usuarios, y que determinaron un total de 24 tarjetas CRC que describieron las clases codificadas a partir de diecisiete Tareas de ingeniería.
3. Se constató la efectividad y validez de la propuesta a partir de las pruebas realizadas. Se verificó que el sistema mantuviera un correcto funcionamiento y quedaron resueltas todas las no conformidades detectadas. Las pruebas de aceptación validaron que el sistema desarrollado responde y satisface las demandas del cliente.

De forma adicional, el cliente como muestra de su satisfacción emitió un aval, dejando constancia de la utilidad de la herramienta y de que el objetivo de esta investigación fue cumplido de forma satisfactoria. Es decir, se logró el desarrollo un sistema que permite el trabajo con la notación estándar de la asignatura y muestra el procedimiento de solución paso a paso para el cálculo de relaciones de recurrencia en la asignatura de Matemática Discreta II de la carrera de Ingeniería en Ciencias Informáticas.

RECOMENDACIONES

Tomando como base la investigación realizada y la experiencia acumulada durante la realización de este trabajo de diploma, se proponen las siguientes recomendaciones:

- Realizar un estudio de los resultados de la implementación de la herramienta para comprobar si la solución propuesta mejora o no el trabajo con las relaciones de recurrencia.
- Integrar la herramienta realizada con otras herramientas que den solución a los temas que abarca la asignatura Matemática Discreta II, para confeccionar un asistente matemático que sirva de apoyo en todos los temas de la asignatura.
- Confeccionar la ayuda y el manual de usuario para facilitarle el trabajo con el software a las personas.

GLOSARIO DE TÉRMINOS

- **JVM:** Máquina Virtual de Java
- **CRC** (Clase-Responsabilidad-Colaboración)
- **GRASP** (General Responsibility Assignment Software Patterns)
- **XP:** Programación Extrema
- **POO:** Programación Orientada a Objetos
- **OO:** Orientada a Objetos
- **API:** Interfaz de Programación de Aplicaciones
- **IDE:** Entorno de Desarrollo Integrado

BIBLIOGRAFÍA

1. **Alitimón, Arletis Francis. 2014.** Portal web Ministerio de desarrollo de software. La Habana: s.n, 2014.
2. **Alitimón, Arletis Francis y Serrano, Nicole Almenares. 2014.** Portal Web del Ministerio de. La Habana: s.n, 2014.
3. **Beck, K. 1999.** Extreme Programming Explained. s.l: Pearson Education, 1999.
4. **Beck, Kent y Flower, Martin. 2000.** Planning Extreme Programming. 2000. 0201710919.
5. **Calderón, S, & Morales, M. 2000.** Relaciones de recurrencia. Costa Rica: Taller de publicaciones del Instituto Tecnológico de Costa Rica: s.n, 2000.
6. **Castro, C.A.C, Blandón, G.E.T. y Tabares, R. de J.B. 2009.** Método y Entorno Integrado de Desarrollo para el Aprendizaje en Lógica de Programación Orientada por Objetos. Revista Iberoamericana de Sistemas. [En línea] 2009. http://www.iiisci.org/journal/CV$/risci/pdfs/GX926UI.pdf.
7. **Collaguazo. 2016.** Software educativo en el proceso de enseñanza aprendizaje de los instrumentos de medida de precisión en los estudiantes del tercer año de bachillerato técnico de la Unidad Educativa Mushuc Pacari. Quito: s.n, 2016.
8. **De la Torre Llorent, Cesar. 2003.** Guia de Arquitectura N-Capas Orientada al Dominio con .NET: Krasis Consulting.ISBN 978-84936696-3-8. España: s.n, 2003.
9. **Echeverría Rodríguez Roberto, Prieto Ramos Álvaro, Sosa Sánchez Encarna. 2006.** Programación Orientada a Objetos.2006.
10. **Echeverry Tobón, Luis Miguel. 2007.** Caso práctico de la metodología ágil xp al desarrollo de software. s.l: Universidad Tenológica de Pereira, 2007.
11. **Esteller, Victor and Medina, Elsy. 2012.** Procesos de desarrollo de software y materiales educativos computarizados. Universidad de Carabobo: Venezuela, 2012.
12. **Fernández Álvarez, José Paulino. 2004.** Análisis y resolución numérica de un problema inverso en geofísica medioambiental. 2004.
13. **Gamma Erich, R.H, Johnson Ralph, VLissides John. 2003.** Patrones de Diseño.Elementos de software orientado a objetos reutilizables, España: s.n, 2003.
14. **Gamma, Erich, y otros. 1994.** Design Patterns: Elements of Reusable Object-Oriented Software. 1994.
15. **García de Jalón, J, Ignacio Rodríguez. 2001.** Aprenda Matlab. Madrid: s.n, 2001.

16. **García de Jalón, Javier. 2000.** Aprenda Java como si estuviera en primero. [En línea] 2000. http://eva.uci.cu/file.php/136/IP/Bibliografia/Complementaria_espanol/Aprenda_Java_como_si_estuviera_en_primero.pdf.
17. **García Sánchez, Ana María. 2010.** Evaluación de métricas de calidad del software sobre un programa Java. Facultad de Informática, Universidad Complutense de Madrid. 2010. pág. 89, Tesis de Maestría.
18. **García. 2011.** Software Educativo para el aprendizaje creativo del curso "Embriología Comparada" Revista Electrónica Educare. 2011.
19. **González Ramírez, Alberto. 2006.** Metodología de la investigación científica.Pontificia universidad Javeriana.2006.
20. **Gutiérrez, J.J. 2006.** ¿Qué es un framework web? [En línea] 2006. http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.
21. **Hung, R. 2014.** Modelo instruccional con soporte en la Plataforma Virtual moodle para la Enseñanza del Contenido integral definida en la Asignatura Cálculo ii. 2014.
22. **IEEE. 2004.** Guide to the Software Engineering Body of Knowledge (SWEBOK). 2004.
23. **Inteco. 2009.** Metodologías ágiles de desarrollo de software. s.l: Laboratorio Nacional de Calidad del Software, 2009.
24. **Java, Netbeans IDE-2015.** Netbeans IDE-Java. [En línea] 2015. https://netbeans.org/features/java/.
25. **Johnsonbaugh, Richard. 2003.** Matemáticas Discretas Cuarta Edición Volumen 2. 2003.
26. **Junit, A. 2013.** Junit A programmer-oriented testing framework for Java. Junit A programmeroriented testing framework for Java. [En línea] 2013. http://junit.org.
27. **Katrib, M. 2003.** Programación Orientada a Objeto en C++. [En línea] 2003. http://eva.uci.cu/file.php/136/IP/Bibliografia/Complementaria_espanol/Miguel_Katrib_Programacion_Orientada_a_Objetos_en_CPP/CAPI1.pdf.
28. **Labrada. 2011.** El Software Educativo un medio de enseñanza eficiente. 2011.
29. **Lamas, Rodriguez. 2000.** Introducción a la informática educativa. La Habana: s.n, 2000.
30. **Larman, C. 2004.** UML y Patrones. Introducción al análisis y diseño orientado a objetos. La Habana, Félix Varela . 2004.

31. **Larman, Craig. 2005.** UML y Patrones 2da Edición Rational Software Canada . Canada: s.n, 2005.
32. **Letelier, Patricio y Penadés, M.Carmen. 2006.** Métodologías ágiles para el desarrollo de software. Valencia: Universidad Politécnica de Valencia, 2006.
33. **Letelier, Patricio. 2006.** Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). Universidad Politécnica de Valencia: Departamento de Sistemas Informáticos y Computación (DSIC),2006.
34. **M. Gómez, lic.Leopoldo Sebastián. 2002.** Diseño de Interfaces de Usuario. Principios, Prototipos y Heurísticas para Evaluación: Poder Judicial del Neuquén . Argentina: s.n, 2002.
35. **Malfará, Dayvis. 2006.** Testing eXtreme Programming. 2006.
36. **Meyer, B, Sánchez, S, Mora, M.K. y Bermejo. 1999.** Construcción de software orientado a objetos. [En línea] 1999. http://www.sidalc.net/cgi-bin/wxis.exe/?IsisScript=AGRIUAN.xis&method=post&form.
37. **Oktaba, H. 2010.** Introducción a Patrones. Facultad de Ciencias, UNAM : s.n, 2010.
38. **Pinciroli, Lic Fernando. 2007.** Necesidad del empleo de herraminetas estándares en XP. Buenos Aires, Argentina: s.n, 2007.
39. **Pressman, R.S. 2006.** Pressman, R.S. Pressman. [En línea] 2006. http://eva.uci.cu/file.php/158/Documentos/Bibliografia_general/Textos_Basicos/Ediciones_del_Pressman/Pressman_7ma_edicion/04_Chapter_1._Software_and_Software_Eng .
40. **Pressman, Roger S. 2005.** Ingeniería del Software un enfoque práctico. 6th. s.l: Mc Graw Hill, 2005. ISBN 9781430271987.
41. **Ramos Fernández, V. 2009.** Desarrollo de un marco de trabajo para Java/J2EE: plataforma Puzzle . [En línea] 2009. http://e-archivo.uc3m.es/handle/10016/6231.
42. **Ramos. 2004.** Los medios de enseñanza,clasificación, selección y aplicación. 2004.
43. **Rizo, Labañino. 2005.** Colección El Navegante. La Habana: s.n, 2005.
44. **Rodríguez, Corbea Maite. 2007.** La metodología Xp aplicable al desarrollo del software en Cuba. 2007.
45. **Rojas, S.G. y Díaz, L.O. 2003.** Java a Tope: J2me (java 2 Micro Edition). [En línea] 2003.

http://books.google.com/books?hl=es&lr=&id=USaAQ0hHQWIC&oi=fnd&pg=PR5&dq=M%C3%A1quina+virtual+de+java+.

46. **Sala, J.J.R. 2003. 2015.** Introducción a la programación.pdf. [En línea] 2015. http://eva.uci.cu/file.php/136/IP/Bibliografia/Complementaria_espanol/Introduccion_a_la_programacion.pdf.
47. **Sanchez, Iriarte. 1999.** Construyendo y aprendiendo con el computador. Integración de medios interactivos para la capacitación de profesores en informática educativa. VIII Congreso Nacional de Informática Educativa. 1999.
48. **Sanchez, Marcelo. 2013.** Diseño y aplicación de un software educativo para desarrollar destrezas con criterio de desempeño del área de Matemática en los estudiantes de quinto año de Educación General Básica de la Unidad Educativa Cristiana Emanuel. 2013.
49. **Sharma, Ritesh. 2014.** Code Metrics in Visual Studio. C# Corner. [En línea] 13 de Octubre de 2014. [Citado el: 18 de Mayo de 2015.] www.c-sharpcorner.com/UploadFile/78607b/code-metrics-in-visual-studio.
50. **Castañeda Porras P**,Quintero Silverio A. y Hernández Vargas,E.2008.Asistente matemático.Herramienta Necesaria en la enseñanza de la matemática.Comité Latinoamericano de Matemática Educativa.[En línea] http://funes.uniandes.edu.co/5090/1/Casta%C3%B1AsistenteALME2008.pdf.
51. **Sommerville, Ian. 2005.** Ingeniería del software. Séptima. Madrid : Pearson Educación, S.A, 2005. ISBN: 84-7829-074-5.
52. **Toctaquiza, Marcelo y Sánchez. 2013.** Diseño y aplicación de un software educativo para desarrollar destrezas con criterio de desempeño del área de Matemática en los estudiantes de quinto año de Educación General Básica de la Unidad Educativa Cristiana Emanuel. 2013.
53. **Trochez, José Luis Orozco. 2013.** timatemáticas. [En línea] 2013. https://sites.google.com/site/timatematicas/home/asistentes-matematicos.
54. **Yang, D. 2010.** Java persistence with JPA. [En línea] 2010. http://dl.acm.org/citation.cfm?id=1841782.