TECNOLÓGICO DE MONTERREY®

Campus Ciudad de México

Escuela de Graduados en Ingeniería y Arquitectura

Maestría en Ciencias de la Computación

"*Definición y control de pruebas aplicadas a un modelo de ciclo de vida de prototipo*"

Autor:
René Pérez Espinosa

Director de la tesis:
Dr. Agustín F. Gutiérrez Tornés

Marzo 2008

"DEFINICIÓN Y CONTROL DE PRUEBAS APLICADAS A UN MODELO DE CICLO DE VIDA DE PROTOTIPO"

por

René Pérez Espinosa

Ha sido aprobada

Marzo del 2008

APROBADA POR LA COMISIÓN DE TESIS:

_____
Dr. Agustín Francisco Gutiérrez Tornes
Asesor

_____
Dr. César Augusto Coutiño Gómez
Sinodal

_____
Dr. Bárbaro Jorge Ferro Castro
Sinodal

ACEPTADA:

_____
Dr. José Martín Molina Espinosa
Director del programa de la
Maestría en Ciencias de la
Computación

Dedicatoria

¿Qué es lo que fue?
Lo mismo que será.
¿Qué es lo que ha sido hecho?
Lo mismo que se hará;
y nada hay nuevo debajo del sol.
(Eclesiastés 1:9)

Gracias a mi señor Jesucristo por estar y llenar mi corazón, a Dios todopoderoso por cuidar de mi vida y a su Espíritu por darme fortaleza.

Este trabajo se lo dedico a mi Esposa Elvia, gracias por animarme a iniciar este proyecto y por ayudarme a concluirlo.

También dedico este trabajo a mis padres Efrén e Hilaria, a mis hermanas Isabel, Yolanda, Laura, a mis hermanos Rubén, Efrén y Jorge, gracias por sus buenos deseos y apoyo.

Agradecimientos

Mi más sincera gratitud a mi asesor de tesis el Dr. Agustín Francisco Gutiérrez Tornés, por su paciencia, su entusiasmo, por su conocimiento y tiempo dedicado para realizar este trabajo.

También agradezco al Ing. Juan Daniel Corte García, por todo el apoyo que me brindó durante el desarrollo de la tesis.

Mi agradecimiento a Nykolas Bernal Henao por su ayuda para mejorar la redacción de este documento.

Quiero agradecer a la Dra. Patricia Rayón Villela, por su valiosa ayuda en todo el proceso administrativo para concluir la tesis.

También quiero agradecer a los sinodales Dr. César Augusto Coutiño Gómez y Dr. Jorge Barbaro Ferro Castro, por su revisión y observaciones realizadas para este trabajo.

Le agradezco al Dr. David Ernesto Salinas Navarro sus comentarios y observaciones para mejorar este trabajo.

Resumen

Esta investigación tiene como finalidad, establecer un método de pruebas aplicable a un modelo de prototipo, el cual presenta ventajas muy útiles en el desarrollo de software, como lo es: menor tiempo de desarrollo, una mejor definición de requerimientos, menor inversión de recursos respecto de otros modelos entre otras cosas.

Dado que la corrección de fallas en el software es un proceso costoso y en el caso del modelo de prototipo, proporciona una reducida preparación ante las fallas de software. El establecer un método de prueba tiene por objetivos, aminorar los costos de producción de software y proveer de mayor calidad en un producto de software. Esto se logra por medio de la detección temprana de errores, con mecanismos de planeación, diseño y ejecución de pruebas adecuadas al modelo.

Adicionalmente se crea un prototipo de herramienta de software que permite realizar un control de la definición de pruebas y su aplicación dentro del ciclo de vida del software, así como automatizar algunas de las actividades en este proceso.

Para el realizar este trabajo se hace uso de los modelos de desarrollo de software en V y el modelo de prototipo incremental, estándares relacionados con la elaboración de pruebas de software e ingeniería de calidad de software.

Contenido

Dedicatoria ..... *i*

Agradecimientos ..... *ii*

Resumen ..... *iii*

Lista de tablas ..... *vi*

Lista de figuras ..... *vii*

1 Introducción *1*

1.1 Antecedentes 2
1.2 Definición del problema 4
1.3 Objetivo de la investigación 6
1.4 Justificación 7
1.5 Hipótesis 7
1.6 Hipótesis nula 8
1.7 Estructura del documento 8

2 Marco teórico *10*

2.1 Metodología de desarrollo de software 10
2.2 Proceso de desarrollo de software 11
2.3 Modelos de desarrollo de software basado en prototipos 15
2.4 Generalidades de pruebas de software 19
2.5 Factores a tomar en cuenta para las pruebas 23
2.6 Técnicas de pruebas de software 24
2.7 Selección de métricas 30
2.8 Análisis del marco teórico 34

3 Generación del modelo de pruebas para prototipo incremental *39*

3.1 Desarrollo del modelo 39
3.1 Plan general de prueba 42
3.2 Pruebas de iteración 43
3.3 Prueba de integración de iteración 44
3.4 Prueba unitaria en la iteración 45
3.5 Pruebas de integración general 46
3.6 Pruebas de sistema 48
3.7 Pruebas de aceptación 49
3.8 Uso de métricas 50
3.9 Consideraciones finales 51

4 Sistema para la planeación de pruebas *52*

4.1 Análisis 52
4.2 Diseño 57
4.3 Construcción, interfaces y resultados 59
4.4 Consideraciones finales 62

5 Pruebas de prototipo *64*

5.1 Descripción de la aplicación a probar 64
5.2 Resultados de las pruebas realizadas 68
5.3 Resultados con el modelo generado 73
5.4 Impacto de defectos en la calidad del software 80
5.4 Consideraciones finales 82

Conclusiones *84*
Contribuciones *85*
Limitaciones *86*
Trabajo futuro *86*
Bibliografía *87*
Apéndice *91*
A Descripción de Propiedades de Calidad 91
Glosario *99*

Lista de tablas

2.1 Ventajas y desventajas del enfoque de caja blanca 29
2.2 Ventajas y desventajas del enfoque de caja negra 30
2.3 Propiedades consideradas por el marco de calidad 32
2.4 Relación entre atributos de calidad y las propiedades que conducen a la calidad 33
2.5 Resumen de características del modelo en espiral 35
2.6 Resumen de características del modelo de prototipo incremental 35
2.7 Resumen de características del modelo de prototipo evolutivo 35
5.1 Datos de líneas de código por iteración 69
5.2 Aspectos revisados en las iteraciones 70
5.3 Detección de errores en la programación por iteración 70
5.4 Resultados de pruebas de valores por iteración 71
5.5 Resultados de pruebas de interfaz por iteración 71
5.6 Resultados de pruebas de compatibilidad por iteración 72
5.7 Porcentaje de densidad error por iteración 73
5.8 Resultado de las propiedades de corrección por iteración 74
5.9 Resultados de las propiedades estructurales por iteración 74
5.10 Resultados de las propiedades de modularidad por iteración 75
5.11 Resultados de las propiedades descriptivas por iteración 75
5.12 Porcentaje de densidad de error con el modelo generado 78
5.13 Comparación de resultado entre los dos modelos 78
5.14 Relación de defectos y atributos de calidad 81

Lista de figuras

2.1 Modelo de cascada 14
2.2 Modelo V 15
2.3 Modelo orientado a reutilización 16
2.4 Modelo incremental 17
2.5 Modelo en espiral 17
2.6 Modelo evolutivo general 18
2.7 Elementos y relaciones que considera el modelo de calidad de Dromey 31
3.1 Modelo incremental modificado (Modelo W) 40
4.1 Esquema general del prototipo 54
4.2 Diagrama de actividades del prototipo 56
4.3 Estructura de funcionamiento del prototipo 57
4.4 Esquema de la base de datos del prototipo 58
4.5 Interfaz del plan general 60
4.6 Ejemplo de captura de prueba de iteración 60
4.7 Inicio de ejecución de pruebas de bajo nivel 61
4.8 Ejemplo de finalización de pruebas de bajo nivel 61
4.9 Evaluación de defectos localizados al final en iteración 62
5.1 Grafica que muestra las líneas de código por iteración 69
5.2 Gráfica de resultados de tipo de prueba por iteración 72
5.3 Errores detectados por iteración 73
5.4 Gráfica de defectos por tipo 76
5.5 Gráfica de errores detectados con el modelo generado 77
5.6 Porcentaje de defectos con antecedentes encontrados en la última iteración 79
5.7 Errores generados en las iteraciones y que permanecieron 79
5.8 Incidencia de defectos en la última iteración 80
5.9 Gráfica de porcentaje de incidencias en atributos de calidad 81

Capítulo 1

1 Introducción

Actualmente el software está inmerso en diversas áreas de la actividad humana, su importancia provoca una mayor demanda en su producción, además de exigencias en calidad y eficacia en los productos de software. Esto implica una necesidad de herramientas, para el desarrollo de software, como lo es el uso de procedimientos y metodologías que permitan cumplir con estas exigencias.

En este sentido, es importante destacar que se consigue el éxito en la construcción de software, siempre y cuando se cumplan los requerimientos, en un tiempo de desarrollo y con recursos tanto humanos como materiales adecuados al entorno del desarrollo, pero sin que los costos de construcción se excedan de manera que sea incosteable un proyecto.

En lo referente a los costos, una de las etapas que representa mayor inversión económica es la de pruebas, algunos autores indican que requiere alrededor de la mitad de los costos totales de desarrollo[1]. Además, esta etapa tiene una gran importancia ya que dependiendo de la efectividad en su realización, tendrá como consecuencia la validación de requerimientos, funcionalidad y ejecución del software final.

Por otra parte, de entre los diferentes modelos de desarrollo de software uno de los que permite tener una mayor eficacia en la definición de requerimientos es el de prototipo, lo cual lleva a generar un desarrollo de software que se apegue en gran medida a las necesidades de la

[1] Wagner, S. and T. Seifert, Software quality economics for defect-detection techniques using failure prediction in Proceedings of the third workshop on Software quality 2005 ACM Press: St. Louis, Missouri pp. 1

aplicación, esta característica entre otras, hace que el modelo basado en prototipos tenga un mayor uso en diversas ramas de la informática.

En el presente trabajo se conjuntan estos elementos: el desarrollo basado en un modelo de prototipos y la definición de pruebas, se presentará un modelo que integre estos elementos, para generar un método de definición de pruebas en las diferentes iteraciones del prototipo, con la finalidad de obtener un producto de software que cumpla con los requerimientos.

1.1 Antecedentes

La informática ha experimentado cambios, debido a la necesidad de adaptarse a las condiciones y recursos del momento. Con el paso de los años, la información aumenta en volumen y diversidad, la tecnología se desarrolla con mayor velocidad y los alcances de aplicación de software aumentan, haciéndolo llegar a usuarios que previamente no lo utilizaban.

Las condiciones cambiantes en los factores relacionados con el software, han propiciado un constante cambio en las metodologías que resuelven la problemática relacionadas con la construcción de software, lo cual ha llevado a la industria a aplicar diversos paradigmas que mejoren las técnicas previamente utilizadas. Actualmente, se dispone de varias metodologías que atacan con mayor efectividad ciertas problemáticas, atributos, características y funcionamiento de las aplicaciones, además de adaptarse y apegarse a la tecnología disponible.

Otro factor importante es la aparición de Internet y su actual evolución, ha provocado que el acceso a la información tenga mayor demanda, los usuarios directos e indirectos de los servicios proporcionados en Internet, crecen año con año y la diversidad de los mismos es mayor cada vez.

Todos estos factores impactan en las técnicas para la construcción de software, se requiere del uso de metodologías diferentes a las tradicionales o bien, una adaptación de las existentes, en el último de los casos, la creación de nuevos procedimientos de desarrollo. Existen varios modelos que manejan de distinta forma la manera en que se desarrolla el software, como son: lineal, cascada, espiral, prototipos, modelo V, orientado a objetos, entre otros.

Dichos modelos, ayudan a los desarrolladores a cubrir ciertos aspectos en el proceso de elaboración de software y han surgido conforme la informática evoluciona, según los factores y características de demanda del momento. Pero el crecimiento tan vasto de las aplicaciones de cómputo fuerza a la industria de software a que se desarrolle en menor tiempo aplicaciones de alta complejidad, de manera que cubran las demandas de servicios o de tecnologías.

El seguimiento o aplicación de un modelo, no necesariamente resuelve la problemática específica del desarrollo de software, en algunos casos es necesario realizar adecuaciones que cubran algún aspecto no considerado en la metodología y si dicha adecuación contribuye a una mayor calidad en el producto final, entonces es importante integrarla de manera sistemática a la metodología, generando con esto una modificación y adaptación de acuerdo a las condiciones del momento.

El inicio para construir un programa de software es recavar información; detallada, clara, sin ambigüedades, lo mas completa y consistente posible. En donde se va a aplicar, que datos va a manejar, los procesos a realizar, la tecnología requerida y de que manera se utilizará. Independientemente de la metodología aplicada, este primer paso es indispensable, por lo que la recolección y análisis de requerimientos se convierte en un punto crítico, ya que incidirá en las directrices a seguir, para el desarrollo de software (metodología, técnicas a utilizar, lenguaje de programación, presupuesto, personal, etc.).

Si se tiene información precisa de los requerimientos, hay una mayor probabilidad de plantear una solución, en un periodo con presupuesto adecuado y cumplir con las condiciones de calidad. En consecuencia se provee de un producto final, que se ajusta a las expectativas y necesidades del usuario final, minimizando el proceso de mantenimiento y corrección de errores, debido a la mejor comprensión del funcionamiento del software.

Dentro de la metodología de desarrollo de software, uno de los modelos que permite definir de forma más precisa las especificaciones y requerimientos es el modelo de prototipo, también es flexible respecto de cambios o adecuaciones en los requerimientos una vez iniciado el desarrollo y debido a que el software se construye en incrementos. Es factible aplicar pruebas, que permitan la detección de errores en etapas tempranas del desarrollo. Lo cual permite establecer un control de calidad para detectar y corregir errores antes de la implantación, además de tener la posibilidad de realizar un seguimiento detallado en la aplicación de pruebas.

1.2 Definición del problema

En la industria de software, se menciona que en lo referente a la calidad, ésta no se puede medir antes de que el producto sea liberado y utilizado por los clientes. En este caso, la calidad se calcula usando el número de fallas encontradas por los clientes, debido a que esta información se conoce en etapas muy tardías del proceso, las acciones correctivas tienden a ser muy caras.[2]

Como se menciona en el apartado anterior, existen diversos modelos de desarrollo, cada uno con características particulares y que tienen ciertas ventajas en determinadas condiciones, sin

[2] Nagappan, N., et al., *Early estimation of software quality using in-process testing metrics: a controlled case study* in *Proceedings of the third workshop on Software quality* 2005 ACM Press: St. Louis, Missouri. p. 1

embargo, éstos dependen en gran medida, de la claridad (entre otras cosas) de la definición de requerimientos como paso inicial y persiguen llegar a un buen final (un desarrollo que refleje y cumpla completamente con los requerimientos), que además minimice al máximo los errores.

Un modelo que permite establecer de forma más precisa los requerimientos es el de prototipo, lo cual como ya se mencionó es un elemento crítico, esta ventaja permite un mejor entendimiento del software a desarrollar, sin embargo una de sus desventajas es el poco énfasis en las pruebas y por tanto la propensión a fallas en el manejo de contingencias del producto final[3]. Además se puede mencionar que el modelo permite generar un software funcional con las partes esenciales del diseño desde el inicio, lo cual provee de una visión más clara de las características del desarrollo y del análisis para su solución conforme se avanza en el proyecto.

A pesar de las prerrogativas que este modelo aporta, la falta de planeación de pruebas impide aprovechar al máximo sus ventajas ya que se dispone de un producto de software que se ajusta en gran medida a los requerimientos del usuario, pero que puede llegar a demandar un mayor esfuerzo en el mantenimiento.

En el caso de proyectos de largo plazo o grandes, es difícil establecer las etapas necesarias y el nivel de funcionalidad del prototipo. Además existen diversos métodos para la realización de pruebas de software y una selección inadecuada puede arrojar mediciones incorrectas[4]. Si no se tiene un control adecuado de las iteraciones y de las pruebas que se realizan, la calidad del producto puede no ser la deseada. Por ello la necesidad de establecer un control metódico desde

[3] Galin, D. (2004). *Software quality assurance : from theory to implementation*. Harlow, Essex ; New York: Pearson/Addison Wesley. p. 119

[4] Alavi, M., *An assessment of the prototyping approach to information systems development* Commun. ACM 1984 **27** (6): p. 558

el inicio del desarrollo aun cuando en muchos casos la naturaleza de la información generada no sea sustancial en las etapas iniciales.

Para esta investigación se pretende abordar el problema del establecimiento de pruebas en las diversas iteraciones en un modelo de prototipo y el control de las mismas, con el propósito de lograr una detección oportuna de errores y con esto conseguir una mejor calidad del producto . El aminorar las desventajas en el modelo genera en consecuencia una herramienta de construcción de software más eficiente, para aquellas aplicaciones que se pueden resolver con este tipo de modelo.

1.3 Objetivo de la investigación

Basado en un modelo de prototipo, generar un nuevo modelo que incluya pruebas de software en las iteraciones, que permita la detección temprana de errores en el mismo. Se espera que al desarrollar el modelo:

- Se definan procedimientos de prueba incrementales que se puedan aplicar en iteraciones.
- Se desarrollen procedimientos, que ayuden a la planeación de pruebas desde el inicio y hasta la finalización del desarrollo.
- Se genere un prototipo de sistema, para la planeación de pruebas, en base al modelo generado en este trabajo.
- Se aplique el modelo generado en un sistema real, para probar su validez.

1.4 Justificación

Emplear una adecuada técnica de prueba puede aminorar el costo final del producto de software, ya que como algunos autores lo mencionan, se asocia el 50% de la inversión de desarrollo a las pruebas[5], la inadecuada selección o aplicación de técnicas de prueba repercute no solo en el funcionamiento sino en el presupuesto y en el peor de los escenarios, podría llevar a la cancelación del proyecto o a la penalización por incumplimiento.

Un elemento que contribuye en "errores" es la discrepancia en la interpretación de requerimientos entre el equipo de desarrollo, el cliente y el usuario final, por está característica en particular, el modelo de prototipo resulta adecuado, debido a que genera entidades funcionales, susceptibles de representar los elementos esenciales del producto en un corto tiempo. Sin embargo, si al prototipo a pesar de no ser un software terminado no se le aplican técnicas adecuadas de prueba y este sigue utilizándose como modelo o base de un futuro desarrollo, podría acumular errores no detectados cuya localización en etapas futuras resulta costoso, de esta necesidad, es el resultado de esta investigación (para mayor detalle, ver capítulo 2).

1.5 Hipótesis

Aún cuando el objetivo de la investigación se ha establecido, se plantea una hipótesis que nos permitirá guiar la investigación, pero en el alcance de este trabajo no esta la corroboración de la misma. Sin embargo, el presente trabajo quedará como un primer paso en ese sentido, siendo requerida una futura investigación, dicha hipótesis es la siguiente:

---

[5] Wagner, S. and T. Seifert, *Software quality economics for defect-detection techniques using failure prediction* in *Proceedings of the third workshop on Software quality* 2005 ACM Press: St. Louis, Missouri pp. 1

Si se establecen pruebas de software, acordes a cada iteración del modelo de prototipo, se mejorará la eficiencia y la efectividad en la detección de errores y como consecuencia, se podrán aminorar los costos de mantenimiento, además de incrementar la calidad del software.

1.6 Hipótesis nula

Diseñar y ejecutar pruebas para las iteraciones en un modelo de prototipo, no implica una mejora en la calidad del producto, ya que no aumenta el nivel de errores detectados, y por lo tanto, no reduce los costos de mantenimiento.

1.7 Estructura del documento

El documento se desarrolla en 5 capítulos, los cuales a continuación se describen, someramente, para dar un panorama de la integración de la presente investigación.

En el capítulo 2 se describen modelos de desarrollo de software y se selecciona un modelo de prototipo de entre las diferentes variantes existentes que ayude a resolver el problema planteado. También se analizan las estrategias de prueba que se pueden usar para el desarrollo de la investigación.

Una vez conseguido lo anterior, para el planteamiento del capítulo 3 se realiza un análisis del modelo seleccionado y en base a éste se generará un nuevo modelo que mejore los defectos de un modelo de prototipo introduciendo planeación de pruebas e incrementando las mismas.

En el capítulo 4, se describen las características del desarrollo de software que ayude a controlar la información generada al aplicar el modelo creado y se plantea la construcción y desarrollo de un prototipo.

Para el capítulo 5, se aplica el modelo creado a un sistema real para verificar si presenta ventajas respecto a los modelos que se tienen actualmente y se realiza un análisis de los resultados obtenidos.

Finalmente, se plantean las conclusiones derivadas del modelo y las pruebas realizadas, las contribuciones que se pueden desprender de la investigación así como las limitaciones y se deja vislumbrar los trabajos que se pueden realizar en el futuro sobre esta investigación.

Capítulo 2

2 Marco teórico

En este apartado se abordan los elementos a partir de los cuales se realiza esta investigación. En primer lugar se expone la necesidad de utilizar una metodología de desarrollo de software, posteriormente los procesos necesarios para la construcción del software, en particular se describen aquellos que se basan en el uso de prototipos. También se describen algunas características, técnicas y estrategias de prueba, se examina su importancia y finalmente se realiza un análisis para establecer una estrategia para la resolución del problema planteado en esta investigación.

2.1 Metodología de desarrollo de software

Cuando surgió la necesidad de adaptar los sistemas informáticos a las exigencias de demanda en la producción de software, el programador realizaba un levantamiento de las solicitudes de quien necesitaba cierto producto software. Con estos requerimientos iniciaba la tarea de codificar esta actividad que estaba administrada o supervisada sólo por el programador; de esta forma a través de lo que se corregía se detectaban los errores, tanto lógicos provenientes de la codificación, como los generados por requerimientos inadecuados por parte del cliente.

En la década de 1970 se denota una mayor complejidad en el desarrollo de software, por lo que la antigua técnica de codificar y corregir empieza a quedar obsoleta, ya que se basaba en requerimientos ambiguos, inconsistentes y con falta de claridad. El cliente en general aportaba

especificaciones muy generales del producto final, sin tener una idea clara de lo que requería o con expectativas poco asequibles.

El desarrollo de software generado de esta forma, finalizaba cuando se satisfacían las especificaciones, aquellas por las cuales surgió la necesidad del programa y también las que fueron apareciendo.

Esta forma de desarrollo era práctica para proyectos pequeños (donde involucraba máximo dos programadores), además de no ser costosa. Sin embargo, para proyectos más grandes o complejos se tenían grandes desventajas en costos, tiempo de desarrollo y calidad.

Derivado de la problemática antes descrita surgió la necesidad de crear metodologías de desarrollo de software. De manera general se puede decir que una metodología es un conjunto de técnicas que se entrelazan para alcanzar un objetivo.

En el caso particular de metodología de desarrollo de software se involucran normas, estándares y se apoya en otras disciplinas relacionadas con la ingeniería, para la construcción de un proyecto software en forma objetiva, sistemática y confiable[6]. Existe una gran variedad de metodologías, que serán de utilidad dependiendo de la naturaleza, el problema a resolver y los elementos disponibles tanto humanos como materiales.

2.2 Proceso de desarrollo de software

Una vez que se plantea la generación de software para atender alguna necesidad, se requiere de algún proceso que permita su realización. Esto con la finalidad de descomponerlo en sus partes esenciales, entender el problema a resolver y aplicar métodos, que den como resultado un producto que cumpla con los objetivos y metas requeridas.

[6] Rico, David F. *Roi of Software Process Improvement : Metrics for Project Managers and Software Engineers*. Boca Ratón, Florida: J. Ross, 2004.

Para ello se utilizan modelos de desarrollo de software, los cuales ayudan a representar y analizar el problema a resolver. El modelo establece una serie de etapas o procesos a realizar y la sucesión de éstos desde su inicio hasta su finalización se le conoce como ciclo de vida.

La ISO (*International Organization for Standardization*), en su norma 12207[7] define al ciclo de vida de un software como un marco de referencia que contiene las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto software, abarcando desde la definición hasta la finalización de su uso.

Actualmente se dispone de una serie de modelos para sistematizar el desarrollo de software, son guías que establecen diversas etapas (o ciclo de vida) para cubrir determinados aspectos del desarrollo, para asegurar su funcionalidad y cumplimiento de los requerimientos al finalizar el desarrollo.

Dichos modelos se han ido transformando con base en las condiciones tecnológicas y a los diversos paradigmas del momento. En cada una de las etapas del ciclo de vida, se pueden establecer una serie de objetivos, tareas y actividades que lo caracterizan.

Desde un punto de vista general (considerando varios tipos de modelos de desarrollo de software), el ciclo de vida de un software tiene tres etapas claramente diferenciadas, las cuales se detallan a continuación:

- Planificación: donde se establece una planeación detallada que guíe la administración del proyecto, temporal y económicamente.
- Implementación: en este rubro se llevan a cabo el conjunto de actividades que componen la realización del producto.

[7]"Industry Implementation of International Standard Iso/Iec 12207: 1995. (Iso/Iec 12207 Standard for Information Technology - Software Life Cycle Processes - Implementation Considerations." *En IEEE/EIA 12207.2-1997*, 1998.

- Puesta en producción: el proyecto entra en la etapa de definición, se le presenta al cliente o usuario final, sabiendo que funciona correctamente y responde a los requerimientos solicitados en su momento. Este momento es muy importante no sólo por representar la aceptación o no del proyecto, sino por las múltiples dificultades que suele presentar la puesta en marcha y adecuación del producto final en un ambiente de uso real, alargándose excesivamente en ocasiones y provocando costos no previstos.

La elección de un modelo para un determinado tipo de proyecto es realmente importante al igual que el orden de construcción. Las principales diferencias entre distintos ciclos de vida se pueden dividir en tres grandes visiones:

- El alcance del ciclo de vida, permite conocer hasta dónde se llegará con el proyecto, si es viable el desarrollo, si se harán actualizaciones y las condiciones del mantenimiento.
- La cualidad y cantidad de las etapas del ciclo de vida: dependiendo de la problemática y características a resolver, el proyecto se dividirá en ciertas etapas, según el ciclo de vida a seguir.
- La estructura y la sucesión de las etapas, indica la forma en que se interconectan las etapas que conforman el ciclo de vida, si hay realimentación entre ellas, si se pueden repetir (iteraciones), el flujo de interconexión y la jerarquía de las etapas.

En los distintos modelos existe la posibilidad de tener que volver a retomar una de la etapas anteriores ya sea por nuevos requerimientos o errores, perdiendo tiempo, dinero y esfuerzo. A esto se le llama riesgo y es diferente en cada etapa. Ningún modelo logra evitar los riesgos que pueden aparecer en el desarrollo de un proyecto.

En seguida se describen brevemente, algunos modelos, su ciclo de vida, características principales y la representación gráfica de los mismos.

Modelo de cascada

Este modelo se realiza en etapas donde cada una debe estar completamente terminada antes de iniciar la siguiente, a pesar de que el modelo tiene retroalimentación al final en cualquiera de las etapas; en la práctica es inoperante y costoso debido a que la generación de la documentación y el cambio en cualquiera de las etapas involucra una adecuación a las siguientes. Es un modelo inflexible, debido a que al inicio se debe disponer de forma precisa, los requerimientos y estos no deben cambiar significativamente en el transcurso del desarrollo. Al término del desarrollo es factible descubrir errores de etapas tempranas, por lo que su mantenimiento puede llegar a ser costoso, la figura 2.1 muestra su estructura.

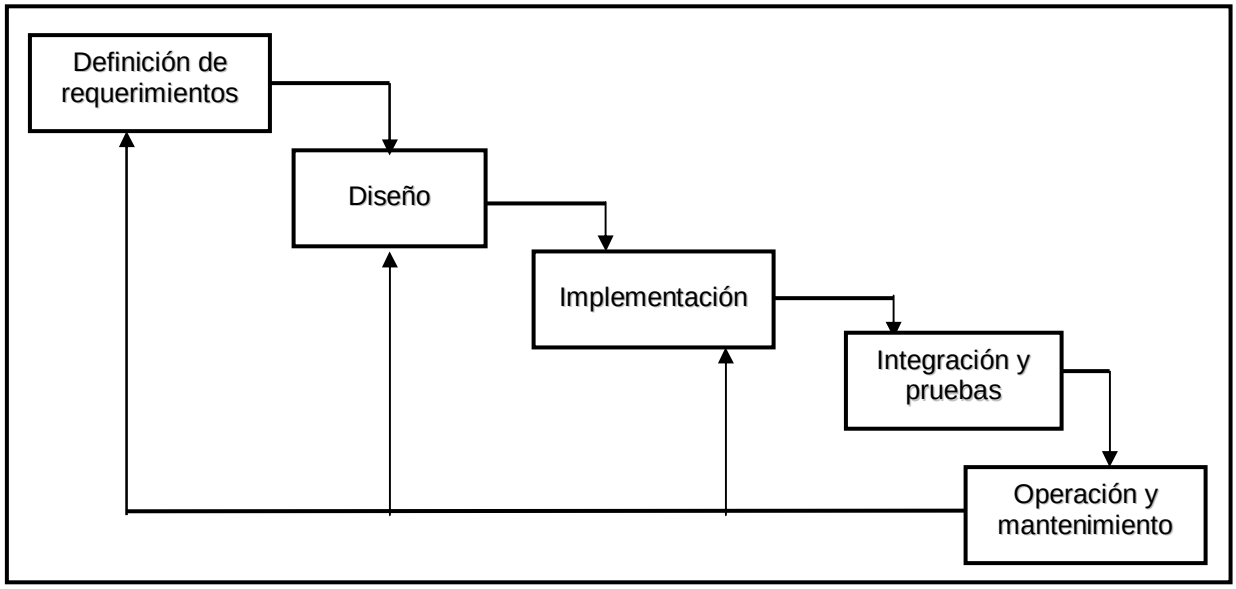

Figura 2.1 Modelo de cascada

Modelo V

Este modelo fue diseñado por Alan Davis, y contiene las mismas etapas que el ciclo de vida en cascada puro. Las ventajas y desventajas de este modelo son las mismas del ciclo anterior, con el agregado de los controles cruzados entre etapas para lograr una mayor corrección, como se

muestra en la figura 2.2. Se puede utilizar este modelo de ciclo de vida en aplicaciones, que si bien son simples (transacciones sobre bases de datos por ejemplo), necesitan una confiabilidad muy alta.

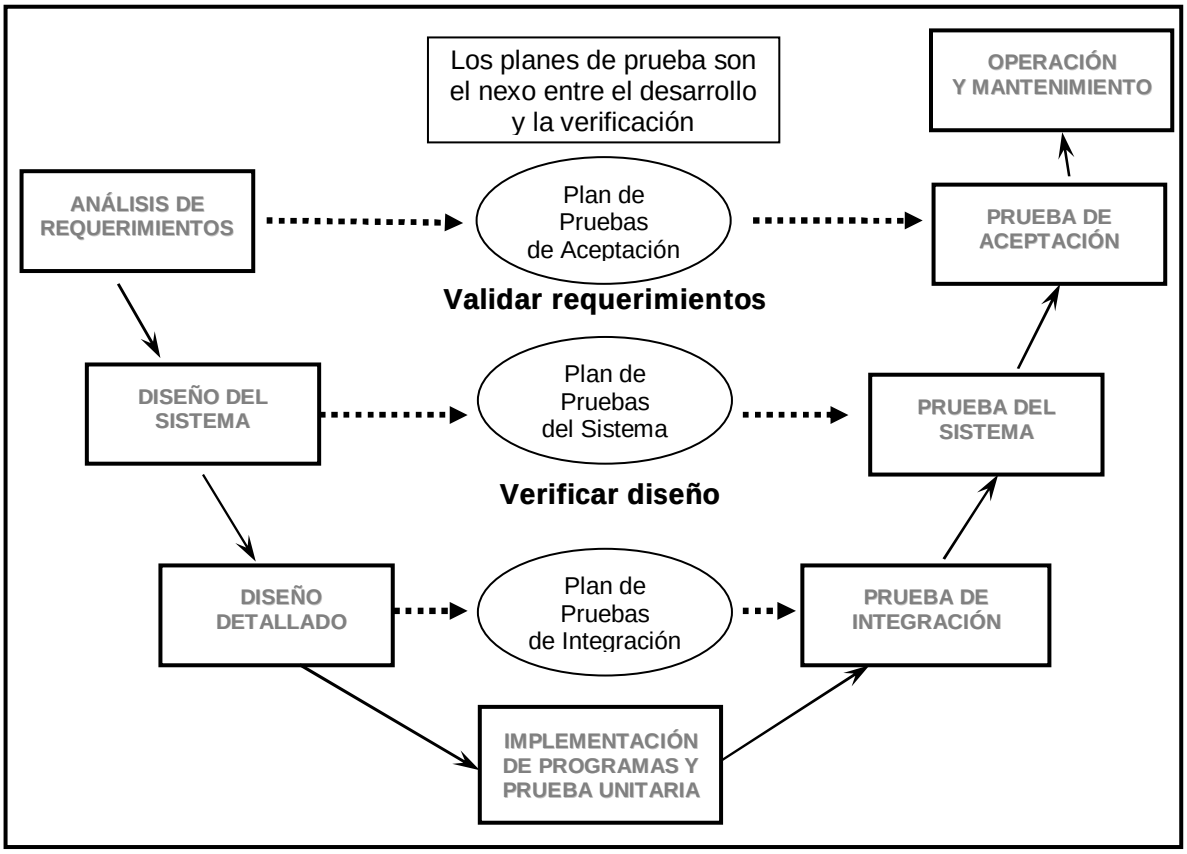

Figura 2.2 Modelo V

2.3 Modelos de desarrollo de software basado en prototipos

En esta sección se muestran otros modelos de ciclo de vida que tienen como característica, identificar etapas que puedan usarse como base (prototipo) para generar un desarrollo posterior con mejoras e incrementos en la funcionalidad necesaria para cumplir los objetivos del desarrollo propuesto.

Desarrollo orientado a reutilización

Este tipo de modelo trata de utilizar componentes que se adecuen a las necesidades del proyecto, que se puedan modificar y en el caso de que no sea posible dicha adecuación, se realiza otro análisis para encontrar un componente adecuado. La principal ventaja de este modelo es la reducción del software a desarrollar, sin embargo es proclive a diferir entre el producto final y las necesidades de la aplicación.

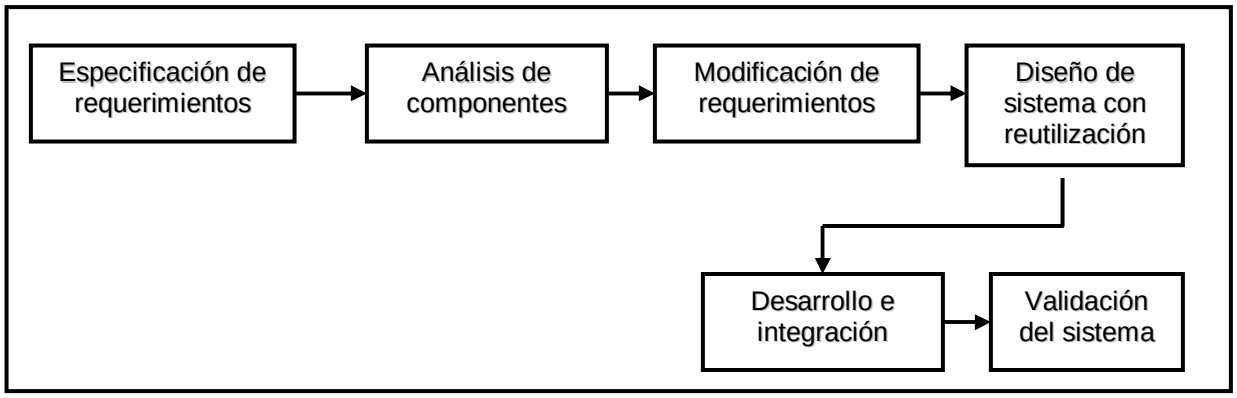

Figura 2.3 Modelo orientado a reutilización

Modelo incremental

Este modelo se basa en la filosofía de construir incrementando funcionalidad en el programa. Es necesario identificar los servicios más importantes, para determinar los incrementos. La ventaja es que los clientes pueden evaluar el desarrollo en cada incremento en vez de esperar su terminación. Existe poco riesgo, ya que se pueden encontrar problemas en los incrementos y corregirlos. Algunos de los problemas con este modelo son: los incrementos que deben ser relativamente pequeños. Y los requerimientos no son definidos en detalle hasta que un incremento sea implementado.

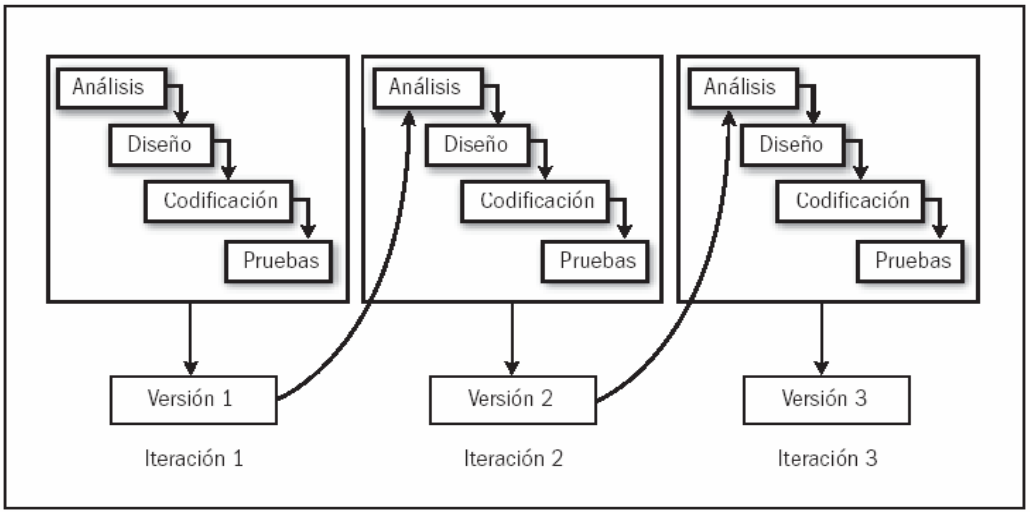

Figura 2.4 Modelo incremental

Modelo en espiral

El modelo en espiral es un proceso evolutivo que acopla la naturaleza iterativa de prototipo con el modelo controlado en cascada. Provee el potencial para el rápido desarrollo para incrementar versiones mas completas de software. Se caracteriza por tomar en cuenta el riesgo en el modelo, pero uno de los problemas, es convencer a los clientes de que la aproximación evolutiva es controlable, se requiere de gran experiencia para tener control de los riesgos de la siguiente etapa, lo cual, si no se lleva a cabo, impacta de manera negativa en la evolución del proceso.

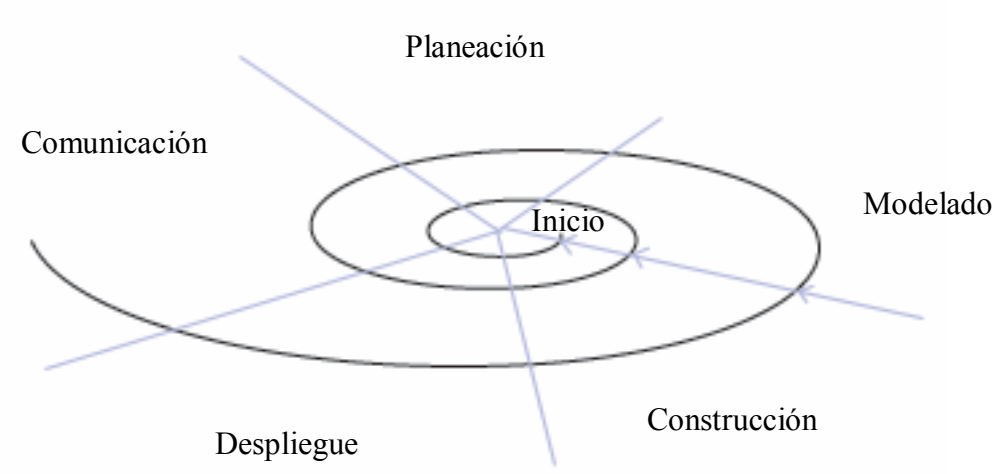

Figura 2.5 Modelo en espiral

Modelo evolutivo

El desarrollo evolutivo esta basado en la idea de desarrollar una implementación inicial, exponiéndola a consideración del usuario y haciendo referencia a través de varias versiones hasta que sea desarrollado un sistema adecuado, en vez de tener una especificación separada, desarrollo y validaciones, las cuales se realizan concurrentemente[8]

Existen dos tipos de desarrollo evolutivo:

- Desarrollo exploratorio
- Prototipo desechable

Un modelo general para ambos casos es el que se muestra en la figura 2.6

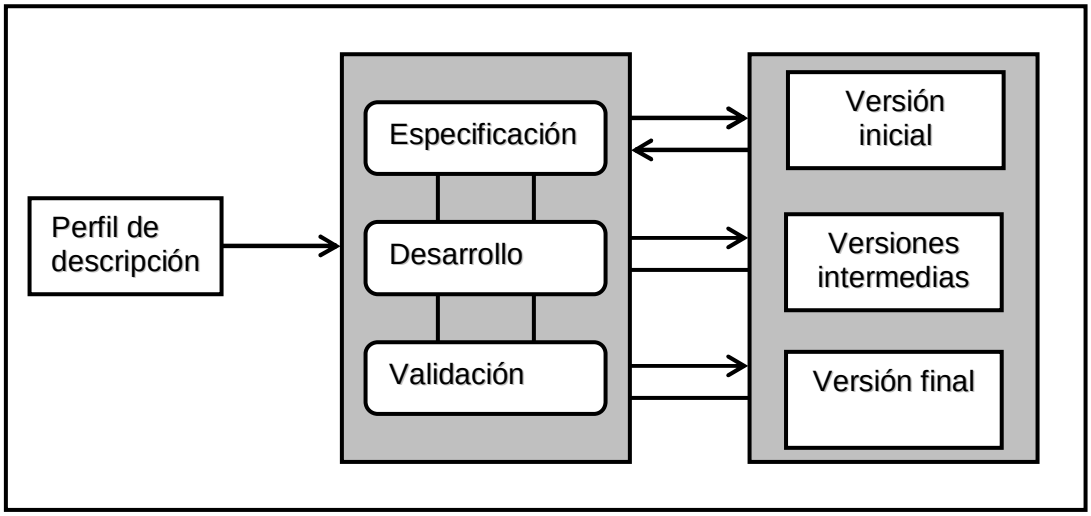

Figura 2.6 Modelo evolutivo general

Este modelo es muy flexible, ya que soporta cambios de requerimientos en el ciclo de vida de desarrollo, además permite desde el inicio del mismo disponer de los elementos esenciales del proyecto, con lo cual puede dar una idea mas precisa del producto final al cliente y

8 Somerville, Ian. 2002. Ingeniería de software. Trad. J. A. D. Torres. 6 ed. Mexico: Pearson Education.

se puede llegar a un producto que se apegue completamente a los requerimientos del sistema. También reduce el riesgo y aumenta la probabilidad de éxito.

Algunas de las desventajas son: que puede dar una idea errónea respecto de los tiempos de desarrollo, si no se maneja adecuadamente el prototipo, puede degenerar.

Algunos de los beneficios de utilizar prototipos son[9] :

- Mejorar la utilidad del sistema
- Un mayor acercamiento del sistema a las necesidades del usuario
- Mejora en la calidad del diseño
- Mejora en el mantenimiento
- Reduce el esfuerzo en el desarrollo

2.4 Generalidades de pruebas de software

Respecto de las pruebas se puede decir que es una actividad muy importante en el área de la ingeniería, ya que tiene como finalidad encontrar defectos, fallas o errores en aquello que se esta construyendo. En la ingeniería de software al igual que en otras ingenierías también se debe probar lo que se construye, en este caso un producto de software, que a pesar de ser un elemento tangible y abstracto a la vez, requiere de técnicas que permitan planear y ejecutar pruebas.

Sin embargo, es prácticamente imposible probar un sistema de forma exhaustiva, ya que las diversas posibilidades y factores involucrados en la ejecución de una aplicación se pueden realizar casi de forma ilimitada. También se puede decir que el tiempo invertido en las pruebas es proporcional al personal dedicado a esta actividad, cantidad y complejidad de las pruebas, en consecuencia los costos asociados solamente a esta actividad, dependen en gran medida de estos factores.

[9] Idem.

Ya que las pruebas implican tiempo y costos considerables, se pretende tener el máximo beneficio del gasto que implica. Por lo que en una aplicación de software cuantos más errores, defectos o fallas se encuentren por unidad de tiempo y costo, más útil será la inversión de esta actividad. Bajo esta perspectiva, el propósito de hacer pruebas es encontrar el mayor número de defectos, con el más alto nivel de severidad posible[10].

Podría resultar paradójico el hecho de que el realizar pruebas, más que demostrar o establecer confianza en el producto, es determinar con firmeza que parte no lo es, en cuyo caso debe ser en aquellas actividades no críticas de la aplicación.

En general se ha tratado a las pruebas como una actividad a posteriori, de detección y no de prevención de problemas en el software, para lo cual intervienen varios factores: sociales (mito de la ausencia de errores en el buen profesional), logísticos (falta de personal o tiempos muy reducidos en entrega de proyectos), desinformación (desconocimiento de procedimientos) y mala comunicación (discrepancias entre los involucrados).

Algunos términos que pueden precisar de mejor manera los aspectos a considerar en lo relativo a las pruebas son los siguientes[11]:

- **Prueba**: una actividad en la cual un sistema o uno de sus componentes se ejecutan en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto.

- **Caso de prueba:** un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular como, por ejemplo, ejecutar un camino concreto de un programa o verificar el cumplimiento de un determinado requisito.

[10] Braude, E. J. (2003). Ingeniería de software : una perspectiva orientada a objetos. México: Alfaomega.

[11] IEEE Guide for the Use of Ieee Standard Dictionary of Measures to Produce Reliable Software." En *IEEE Std 982.2-1988*, 1989.

- **Defecto**: un proceso, definición de datos o una parte de procesamiento incorrectos en un programa.

- **Falla**: la incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados.

- **Error:** tiene varias acepciones:
  1 La diferencia entre un valor calculado, observado o medido y el valor verdadero, especificado o teóricamente correcto.
  2 Instrucción incorrecta de un programa.
  3 Resultado incorrecto.
  4 Una acción humana que conduce a un resultado incorrecto.

En adelante se tomará el término error para englobar los términos defecto y falla, ya que la acepción número uno de la definición de error engloba el hecho de diferir entre un resultado respecto a algo esperado, aunque se respetarán los términos cuando sea necesario.

Retomando el proceso de desarrollo y englobando las actividades, hasta cierto punto comunes, de los modelos de desarrollo se pueden listar las siguientes:

- Análisis de requerimientos de software,
- Diseño de software,
- Métodos de programación,
- Procedimientos de prueba,
- Verificación y validación,
- Configuración del software y
- Aseguramiento de la calidad del software

En estas actividades se puede observar que independientemente del modelo a seguir, un apartado importante, es la fase de pruebas, que valora el correcto funcionamiento de aquello que se está

desarrollando. Por otro lado, se puede decir que un proyecto de software exitoso se deriva de la planeación de un producto a través de un esquema de presupuesto, conociendo las funciones y calidad requeridas.[12]

Con estos argumentos se puede notar la importancia de las pruebas en el desarrollo de software, sin embargo la fase de pruebas se puede entender desde diversas perspectivas y es una etapa que va más allá del hecho de depurar código, en algunas ocasiones se integra la fase de pruebas a la implementación por la situación de revisar el código fuente, lo cual es solo una parte del proceso en esta etapa, ya que la prueba involucra la ejecución del producto de software con la intención de encontrar errores físicos o lógicos.[13]

Durante mucho tiempo la fase de prueba no se consideraba como una disciplina de la ingeniería de software sino hasta la década de los 80[14]. En un enfoque mas amplio, las pruebas tratan de encontrar errores, no probar si un programa es correcto, es prácticamente imposible probar completamente un programa, sobre todo con la magnitud de líneas de código que se manejan actualmente.

Este trabajo se apoya en la siguiente definición de prueba de software, para su desarrollo:

"La prueba de software es un proceso formal llevado a cabo por un equipo especializado en el cual una unidad de software, varias unidades de software integradas o un paquete entero de software es examinado, ejecutando los programas necesarios en un equipo de cómputo. Todas las pruebas asociadas se ejecutan de acuerdo a un procedimiento formalmente aprobado con casos previamente establecidos y/o probados"[15].

[12] Dorfman, Merlin y Richard H. Thayer. 1997. Software Engineering. USA: IEEE Computer Society Press.

[13] Idem.

14 Charette, Robert N. 1986. *Software engineering environments: Concepts and technology*. New York.

[15] Piattini Velthuis, Mario G. *Calidad En El Desarrollo Y Mantenimiento Del Software*. México: Alfaomega, 2003.

2.5 Factores a tomar en cuenta para las pruebas

Ya se ha mencionado la importancia de las pruebas, también es importante considerar que esta actividad debe ser sistemática y una buena planificación permitirá detectar el máximo número y variedad de defectos con el mínimo consumo de tiempo y esfuerzo, las siguientes son algunas recomendaciones que hace G.J. Myers [16] :

- Cada caso de prueba debe definir el resultado de salida esperado. Este resultado esperado es el que se compara con el realmente obtenido de la ejecución en la prueba. Las discrepancias entre ambos (errores) se consideran síntomas de un posible defecto de software.
- El programador debe evitar probar sus propios programas, ya que desea (consiente o inconscientemente) demostrar que funcionan sin problemas.
- Se debe inspeccionar a conciencia el resultado de cada prueba para, así, poder descubrir posibles síntomas de defectos.
- Al generar casos de prueba, se deben incluir tanto datos de entrada válidos y esperados como no válidos e inesperados. Es frecuente observar una tendencia a centrarse en lo esperado y lo valido.
- Las pruebas deben centrarse en dos objetivos :
    - Probar si el software no hace lo que debe hacer.
    - Probar si el software hace lo que no debe hacer, es decir, si provoca efectos secundarios adversos

[16] Idem.

- Se deben evitar los casos desechables, es decir, los no documentados, ya que se puede caer en repeticiones innecesarias.
- No deben hacerse planes de prueba suponiendo que no hay defectos o gran cantidad de ellos y como consecuencia dedicar pocos recursos a las pruebas.
- En la experiencia se observa que la probabilidad de descubrir nuevos defectos en una parte del software es proporcional al número de defectos ya descubierto.

2.6 Técnicas de pruebas de software

Como ya se ha descrito el proceso de prueba de software es una actividad que implica varias consideraciones, por lo que una decisión crítica y primordial es la de establecer la manera en que se realizarán las pruebas. Algunas preguntas que ayudarán a establecer una estrategia adecuada son: [17]

¿Qué componentes deben probarse?
¿De qué manera se deben probar?
¿Qué se espera de la prueba?

Estas preguntas implican la consideración del tiempo a invertir en las pruebas, el cual aumenta proporcionalmente de acuerdo al número de componentes a probar, el personal involucrado y la cantidad de cambios en el software y la complejidad del producto.

Respecto de las pruebas que se pueden realizar en un producto de software, de manera general se pueden clasificar de la siguiente manera:

- Pruebas estáticas
- Pruebas dinámicas

[17] Charette, Robert N. 1986. *Software engineering environments: Concepts and technology*. New York.

- Pruebas de integración

Las pruebas estáticas en general se utilizan para las pruebas unitarias, que revisan principalmente la estructura del código fuente, ayuda a encontrar errores de lógica en el programa y permite revisar las prácticas de programación. Algunas de las limitaciones en las pruebas estáticas son: su difícil aplicación en programas muy grandes y no permite la detección de errores en tiempo real.

Las pruebas dinámicas se enfocan en la ejecución del programa y la comparación de resultados respecto de las especificaciones, dentro de este rubro se puede dividir dos tipos de pruebas dinámicas:

- Funcionales
- Lógicas

Las pruebas funcionales verifican que el programa se ejecute adecuadamente bajo condiciones operacionales típicas, de manera que el proceso realizado de acuerdo a una entrada y la salida generada correspondan a las especificaciones del programa (desde el punto de vista de caja negra).

La prueba lógica evalúa como se ejecuta el programa desde el punto de vista interno (caja blanca), en general se trata de probar las diversas condiciones y posibles valores para éstas, sin embargo esto es difícil de conseguir por la magnitud de posibilidades. Este tipo de prueba también verifica: los datos y el control interno entre las diferentes unidades que componen el programa, el tiempo de ejecución y de respuesta de los procesos, tiempo de uso del procesador, accesos a red, envío y recepción de información de forma remota.

Respecto de las pruebas de integración evalúan y verifican los diversos módulos que componen al sistema, este tipo de pruebas permite detectar errores en las interfaces,

interconexiones, en los tiempos de respuesta totales en el desempeño final del sistema y los estándares de calidad. Dentro de este rubro se tiene otro tipo de prueba que es la prueba de aceptación o certificación que consiste en examinar al sistema en condiciones reales, en un ambiente diferente al de desarrollo y probar desde la instalación, la configuración y la ejecución del producto de software bajo condiciones acordes a las especificaciones en un entorno similar al del uso destinado.

La cantidad, complejidad y diversidad de pruebas a realizar, van muy de la mano con el costo a invertir en esta actividad, el personal disponible y el tiempo en el que se pretenda realizar esta etapa, por lo general es una de las etapas más costosas dentro del ciclo de vida del software. Por lo que es importante considerar una estrategia adecuada en la forma de realizar las pruebas. Algunas propuestas de estrategia son las siguientes:

- Planeación, diseño, ejecución y mantenimiento.[18]
- Decisión para automatizar pruebas, herramienta de prueba, introducción al proceso de pruebas automatizadas, planeación, diseño y desarrollo de pruebas, Ejecución y manejo de pruebas y revisión/valoración.[19]
- Pruebas unitarias, pruebas de Integración, validación de pruebas.[20]
- Planeación de la prueba, prueba de requisitos, pruebas del diseño, pruebas en el desarrollo y, pruebas en la implementación[21]

Estas estrategias de diferentes autores se pueden englobar en tres procesos, la planeación, el diseño y la ejecución. La diferencia notable dentro de estos elementos, es que se aplican en

[18] Goglia, Patricia. 1993. *Testing client/server applications*. USA: QED Publishing group.
[19] Dustin, Elfriede. Jeff Rashka. John Paul. 1999. Automated software testing: introduction, management and performance. Boston: Addison-Wesley.
[20] Pressman, Roger S. 2005. *Software Engineering*.
[21] Hetzel, William C. 1998. *The complete guide to software testing*. USA: John Wiley & Sons Inc.

diferentes momentos en el ciclo de vida del desarrollo de software. Por lo que se puede observar la diversidad en los procesos a seguir, e inclusive algunos autores tratan a la etapa de prueba, como un ciclo de vida por si mismo.

2.6.1 Estrategias para prueba de integración

Existen dos tipos de estrategias para realizar pruebas que verifiquen la integración de módulos o unidades de software en un sistema[22]. Una es la estrategia llamada "big bang" o no incremental, la otra es denominada incremental que a su vez se divide en dos: "bottom up" (descendente) y "top down" (ascendente). En la estrategia "*big bang*" se prueban todos los módulos, lo cual implica una revisión exhaustiva, que como se mencionó previamente se descarta esta posibilidad, lo que nos deja dos posibles estrategias que en seguida se mencionan brevemente:

Estrategia "*bottom up*" consiste en revisar los módulos de más bajo nivel de la estructura de software hasta llegar al de mayor nivel o nivel principal. La ventaja a destacar de esta estrategia es su relativa facilidad de ejecución, su desventaja consiste en la tardanza de poder observar al sistema como un todo.

Estrategia "*top-down*" consiste en iniciar la revisión con el módulo principal y subsecuentemente los módulos de más bajo nivel. La principal ventaja de esta estrategia es la posibilidad de mostrar las funciones del programa en un corto periodo (una vez que se ha revisado el módulo principal), en muchas ocasiones esta característica permite una temprana identificación de errores en el análisis y diseño relativo a los algoritmos y requerimientos funcionales. La principal desventaja es la relativa dificultad en el análisis de resultados.

[22] Piattini Velthuis, Mario G. [et al]. *Analisis Y Diseño Detallado De Aplicaciones Informáticas De Gestión*. México: Alfaomega : Ra-Ma, 2000.

Para este trabajo se utilizará la estrategia "top down" ya que permite visualizar un proyecto como un todo funcional, para realizar las pruebas de software, esta característica es el elemento relevante y en común con el modelo de prototipo, que en sus etapas tempranas están construidos solamente aquellos módulos que son esenciales para mostrar las características mas importantes del proyecto.

2.6.2 Enfoques de prueba

El enfoque de prueba, hace énfasis en ciertos aspectos de la prueba con relación directa a los requerimientos de calidad de forma estructural o funcional, existen tres enfoques caja blanca, caja negra y caja gris como se mencionó en el Subtema 2.6, en seguida se muestra los casos en que es factible utilizar cada enfoque[23].

Enfoque de caja blanca

Cuando se requiere tener un alto grado de certeza en la aplicación de estándares, mejores prácticas, aplicaciones de alta seguridad y críticas en cuanto su funcionamiento, optimización de código e inclusive en la remoción de comentarios (ya sea informativos o líneas de código comentariadas) innecesarios, el enfoque de caja blanca o también llamado estructural es una buena opción. En etapas muy tempranas de desarrollo cuando prácticamente se tienen solo unidades funcionales de software, este enfoque es muy adecuado.

Enfoque de caja negra

Este enfoque se utiliza principalmente, para pruebas de exactitud de datos de salida. Cuando en el proyecto no se pondrá énfasis en la estructura interna sino en la funcionalidad y

[23] Braude, Eric J.. *Ingeniería De Software : Una Perspectiva Orientada a Objetos*. México: Alfaomega, 2003.

comportamiento del sistema, este enfoque es adecuado. A pesar de la desventaja de requerir una gran cantidad de casos de prueba, para poder verificar de forma aceptable la funcionalidad, salida e intercambio de datos en un sistema, existen alternativas que proporcionan un grado adecuado de confiabilidad y reducen en gran medida los casos de uso necesarios, teniéndose como consecuencias, reducción en los costos y optimización en el proceso de prueba.

Enfoque de caja gris

Es una combinación de caja blanca y negra que consiste en una revisión estructural pero limitado a una porción de código que puede ser importante analizar y además, probar el comportamiento de la unidad de software que se este revisando con la finalidad de observar los elementos de diseño y los resultados esperados. En general, este enfoque se utiliza sólo para ciertas unidades cuyo detalle de prueba sea mayor, tanto para la estructura interna como el entorno.

Algunas ventajas y desventajas de cada enfoque que pueden ayudar para tomar una decisión de cual utilizar se muestran en la tabla 2.1 y2.2:

Enfoque de caja blanca	
Ventajas	Desventajas
• Se revisa directamente cada línea de código (en función de las rutas a probar) y a su vez los algoritmos utilizados. • Se puede identificar las líneas de código que no se ejecutan en los casos de pruebas y ejecutar los casos de prueba pertinentes si es necesario. • Permite determinar la calidad de la codificación y el apego a los estándares de codificación	• Se requiere una gran cantidad de recursos para ejecutar este enfoque • No se puede probar el desempeño en términos de disponibilidad (tiempo de respuesta), confiabilidad y otras pruebas relativas a la operación, revisión y factores de transición.

Tabla 2.1 Ventajas y desventajas del enfoque de caja blanca

Enfoque de caja negra	
Ventajas	Desventajas
• Permiten realizar la mayoría de los tipos de pruebas (sin necesidad de usar otro enfoque). • Algunas pruebas, también se pueden realizar con caja blanca, sin embargo requieren menos recursos.	• Es posible que la agregación de errores coincida con una respuesta correcta para un caso de prueba en particular e impida la detección de errores. • Ausencia del control en el alcance, cuando el encargado de realizar la prueba intenta mejorar la línea de alcance, no existen los mecanismos para especificar los parámetros en los casos de prueba para dicha mejora. Por lo anterior, puede no ejecutarse proporciones substanciales de líneas de código. Imposibilita la prueba de calidad de la programación y su apego a estándares.

Tabla 2.2 Ventajas y desventajas del enfoque de caja negra

2.7 Selección de métricas

Una parte importante del desarrollo de software es la evaluación del mismo, para lo cual resultan de gran utilidad las métricas de software, permiten medir características particulares del software. Existe una gran cantidad y diversidad de métricas aplicables a las diferentes etapas del desarrollo de software.

En este trabajo se considera el marco de trabajo de un modelo para la calidad de producto de software propuesto por R. Geoff Dromey[24], el cual además de generar información de defectos

[24] Dromey, R. G. (1995). A model for software product quality. *Software Engineering, IEEE Transactions on*, 21(2), 146-162.

y errores en el software, permite evaluar atributos de calidad, en seguida se explica brevemente las características principales de este marco de trabajo.

Se utiliza un modelo genérico formado por tres entidades principales y sus relaciones, como se muestra en la figura 2.7, considera cuatro relaciones entre las entidades que pueden conducir a la calidad del producto, los componentes en el modelo se consideran como formas estructurales en el software, las relaciones que toma en cuenta son las siguientes:

- Forma estructural → Propiedad de calidad
- Propiedad de calidad → Atributo de calidad
- Atributo de calidad → Propiedad de calidad
- Propiedad de calidad → Forma estructural

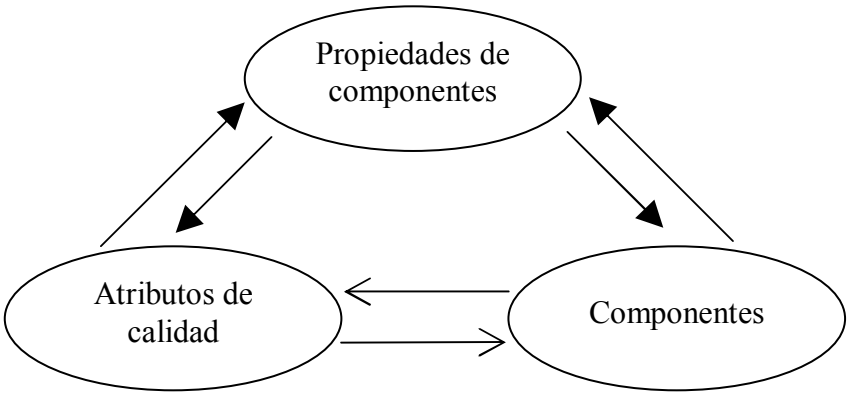

Figura 2.7 Elementos y relaciones que considera el modelo de calidad de Dromey

El modelo se centra en el nivel más bajo de pruebas, ya que pretende identificar que propiedades requieren ser satisfechas, para cada forma estructural y así alcanzar calidad en los atributos de alto nivel. Por tanto, si se viola alguna propiedad que provee de calidad a una forma estructural, el resultado es un defecto en la calidad del producto.

El modelo plantea una serie de propiedades conducentes a la calidad del producto, agrupadas por categorías que se muestran en la tabla 2.3.

Categoría	Propiedad
Corrección (Requerimientos mínimos para corrección)	• Computable • Completa • Asignada • Precisa • Inicializado • Progresivo • Variante • Consistente
Estructural (Problemas de diseño intramodular de bajo nivel)	• Estructurada • Resuelta • Homogénea • Efectiva • No redundante • Directa • Ajustable • Independencia de rango • Utilizada
Modularidad (Problemas de diseño intermodular de alto nivel)	• Parametrizada • Acoplamiento débil • Encapsulado • Cohesivo • Genérico • Abstracto
Descriptiva (varias formas de especificación/documentación)	• Especificada • Documentada • Auto descriptiva

Tabla 2.3 Propiedades consideradas por el marco de calidad

Un vez que se han considerado las formas estructurales, propiedades que propician calidad y un conjunto de atributos de alto nivel, entonces se puede utilizar el modelo de calidad de software, el cual se basa en el siguiente teorema[25]:

"Si cada propiedad que conduce a la calidad, que se asocia con una forma estructural en particular es satisfecha, cuando esa forma estructural se usa en el programa, entonces la forma estructural contribuirá a la eliminación de defectos en el software".

[25] Idem.

Finalmente se relacionan las propiedades antes descritas, con su incidencia en los atributos de calidad de software, como se muestra en la tabla 2.4

	Propiedades de Corrección								Propiedades Estructurales								
	Computable	Completa	Asignada	Precisa	Inicializado	Progresivo	Variante	Consistente	Estructurada	Resuelta	Homogénea	Efectiva	No redundante	Directa	Ajustable	Independencia de rango	Utilizada
Funcionalidad	X	X	X	X	X	X	X	X	X								
Confiabilidad	X	X	X	X	X	X	X	X	X								
Usabilidad		X						X				X					
Eficiencia										X		X	X	X			X
Mantenibilidad		X			X	X	X	X	X	X	X	X	X	X	X	X	X
Portabilidad								X							X		
Reutilización								X							X	X	

	Propiedades de Modularidad						Propiedades Descriptivas		
	Parametrizada	Acoplamiento débil	Encapsulado	Cohesivo	Genérico	Abstracto	Especificada	Documentada	Auto descriptiva
Funcionalidad							X		
Confiabilidad		X	X				X		
Usabilidad							X	X	X
Eficiencia									
Mantenibilidad	X	X	X	X	X	X	X	X	X
Portabilidad	X	X	X	X	X		X	X	X
Reutilización	X	X	X	X	X	X	X	X	X

Tabla 2.4 Relación entre atributos de calidad y las propiedades que conducen a la calidad

Con estos elementos y una serie de recomendaciones relativas a cada tipo de propiedad por parte del autor del modelo, se pueden realizar pruebas de bajo nivel para disminuir en gran medida defectos que se propaguen y así proporcionar mayor calidad al software.

2.8 Análisis del marco teórico

Como menciona Alvin Toffler, en la actualidad los medios de producción ya no se basan totalmente en la industria y en el esfuerzo humano, sino en las capacidades intelectuales, algunos la llaman, la era del conocimiento la que es fuertemente conducida por los elementos tecnológicos. En la actualidad la información es productora por sí misma de riqueza y de ahí la importancia en el desarrollo de software[26].

La ingeniería de software, en general, establece el uso de principios de ingeniería para concebir productos de software con calidad, que optimicen los recursos de hardware, que eficiente los procesos para los que fueron creados, que se puedan mantener relativamente a bajo costo y escalables entre otras cosas.

Dado que para la creación de software se requiere una metodología de desarrollo y por tanto un modelo a seguir, en primer lugar se elige un modelo de desarrollo. Como ya se mencionó, en el caso del modelo de prototipo encontramos que puede sustentar una definición adecuada de requerimientos sobre todo en esquemas donde no se tiene una idea clara de los procesos a resolver y sus características o cuando no hay un adecuado entendimiento entre el cliente y el equipo de desarrollo,

En las tablas 2.5 a 2.7 se muestra un resumen de las ventajas y desventajas de las variaciones del modelo de prototipo. Dado que el problema a resolver es una mejor detección de errores en el desarrollo de software a un costo adecuado y en un tiempo menor, y los diferentes modelos de prototipo no proporcionan una solución es necesario crear un nuevo planteamiento como el que se presenta en este trabajo.

[26] Toffler, Alvin. *La Tercera Ola*. Barcelona: Plaza y Janés, 1999.

Modelo en Espiral	
Ventajas	Desventajas
– El producto avanza solucionando riesgos en cada iteración. – El producto termina con todos los riesgos resueltos. – Se pueden incluir otros métodos de desarrollo en las iteraciones. – A medida que el costo aumenta, los riesgos se reducen. – Se tienen puntos de control en cada interacción.	– Es complicado – Requiere de mucha administración – Difícil de definir los objetivos, metas que indiquen que podemos avanzar al siguiente ciclo. – Se puede caer en un desarrollo de nunca acabar. – Aumento en el costo de desarrollo.

Tabla 2.5 Resumen de características del modelo en espiral

Modelo incremental	
Ventajas	Desventajas
– Cada etapa agrega funcionalidad. – Permite obtener un producto más rápido. – Reduce riesgos al observar el progreso y retroalimentación. – El progreso se puede medir en periodos cortos de tiempo. – Ayuda a un mejor entendimiento del problema y su solución.	– Puede dar una falsa perspectiva al usuario sobre la velocidad del desarrollo. – Es proclive a propagar errores entre iteraciones. – Permite redefinición de requerimientos

Tabla 2.6 Resumen de características del modelo de prototipo incremental

Modelo evolutivo	
Ventajas	Desventajas
– Permite precisar requerimientos que no son claros. – Es flexible ante cambios de requerimientos. – Gran interacción con el cliente – Desarrollo de versiones mejoradas en base a retroalimentación. – Gran énfasis en cumplir con los requerimientos.	– Es difícil realizar una planeación efectiva. – Se puede convertir en el producto final aún y cuando no cumpla con todos los requerimientos. – Demasiados cambios puede conducir a errores no previstos. – Puede degenerar el proyecto ante grandes cambios en los requerimientos.

Tabla 2.7 Resumen de características del modelo de prototipo evolutivo

De los diversos tipos de prototipo, el que permite obtener un producto con mayor rapidez y mejores condiciones de desarrollo es el modelo de prototipo incremental (ver sección 2.3 y capítulo 3). Además, permite la creación de prototipos funcionales, que podrían utilizarse para realizar pruebas o estudios de factibilidad en ambientes reales y asegurar, tanto la funcionalidad como el éxito de un proyecto en ambientes tan variados como Internet, aplicaciones inalámbricas, telefonía celular, etc., una vez establecidos los alcances y ajustes que se deben realizar en el proyecto, el prototipo tiende hacia la aplicación real. Por tanto se utilizará este modelo para el desarrollo de esta investigación.

Como en otras disciplinas de la ingeniería, efectuar pruebas en un producto de software para asegurar su efectividad y calidad, implica realizar un proceso de prueba adecuado, esto ayuda a la detección de errores o fallas, elementos que no maneja adecuadamente el modelo de prototipo incremental.

Como se menciona en la sección 2.2 de este capítulo, el modelo V proporciona una confiabilidad muy alta gracias a mecanismos de corrección. Dicha característica se puede aprovechar para integrar mejores procesos de prueba en el modelo de prototipo incremental, por lo que también se tomará en cuenta este modelo.

El punto central de este trabajo, es la integración de pruebas en el proceso iterativo del modelo de prototipo, para dar un mayor soporte a la detección de errores y como resultado contar con un modelo que en su aplicación, establezca criterios específicos que contemplen la mayoría de los elementos involucrados con el software que se desarrolla (hardware, sistema operativo, arquitectura, etc.), durante cada iteración y así disponer de un prototipo, en cada una de ellas, que haya sido probado adecuadamente como si fuera el producto de software final.

Otra ventaja a mencionar es la de integrar un mejor proceso de pruebas en la aplicación del modelo, es la de tener un control en la información generada para éste proceso, dicha información permite tener puntos de referencia para el desarrollador o desarrolladores, además, proporciona un nivel de flexibilidad en el seguimiento de errores en las etapas de integración.

En cierta forma se tendrían "fotografías" históricas de las pruebas realizadas, los elementos probados y los resultados, con lo cual se puede realizar un seguimiento para ubicar una falla o el impacto en un cambio de especificación y hasta la inferencia de un error no considerado. En el proceso de mantenimiento el disponer de un detalle más especifico de las pruebas ejecutadas, componentes revisados, casos de prueba, etcétera, es de gran ayuda para la resolución de fallas.

Estrategia a seguir

Para lograr el objetivo planteado, se tomarán en cuenta las ventajas de los modelos de prototipo incremental y V, se revisará la estructura de diseño y las características de las etapas que los componen, para así generar un nuevo modelo que mantenga sus ventajas, pero que además permita establecer mecanismos de prueba, para una mayor detección de errores y que ayude a una mejor planeación.

Además de generar un nuevo modelo también se determinará que pruebas son adecuadas a éste, en que etapas es conveniente realizarlas y que consideraciones se deben hacer, por lo que se hará uso de un marco de calidad de software, que permite evaluar y localizar errores en el proceso de desarrollo, además de identificar defectos en atributos de calidad.

Para comprobar lo anterior, se aplicará el nuevo modelo en un sistema real para observar su funcionamiento y verificar que ayude a una mayor detección de errores. También se desarrollará un prototipo de software que ayude a la automatización de las tareas de planeación para el modelo que se obtenga y pueda servir de referencia para trabajos futuros.

Por lo anterior, en el siguiente capítulo se realiza el análisis de los elementos a utilizar de los modelos seleccionados, de que manera se puede realizar una integración de sus características para llegar a un nuevo modelo de desarrollo.

Capítulo 3

3 Generación del modelo de pruebas para prototipo incremental

En este capítulo se desarrollan mejoras en el modelo de prototipo incremental, tomando en cuenta las características del Modelo V. Se pretende aprovechar las ventajas de ambos modelos, tanto en la perspectiva de previsión de pruebas para cada etapa de desarrollo, como en la forma de desarrollo de un proyecto.

Ya que las ventajas del modelo de prototipo son el factor principal por el que se seleccionó, se mantendrá la estructura de éste modelo y se modificará la etapa de prueba, agregando una estructura de pruebas que involucre las etapas de desarrollo en la iteración.

3.1 Desarrollo del modelo

Como se observa en el diagrama de prototipo incremental de la figura 2.2, para cada iteración existe una etapa final que es la de prueba, sin embargo no hay una especificación del tipo de pruebas a realizar a diferencia del Modelo V figura 2.4, que toma en cuenta en todo momento del desarrollo, pruebas específicas para cada etapa.

Así que, retomando la estructura del Modelo V e integrando algunas de sus características al modelo de prototipo se produce el diagrama de la figura 3.1, que en lo posterior se le llamará modelo W. En este nuevo modelo, se enfatiza la planeación de pruebas durante el desarrollo de cada etapa de la iteración, con una jerarquía de prueba en paralelo.

Bajo este modelo, las pruebas también evolucionan o se incrementan conforme se desarrollan las iteraciones, toma en cuenta, los resultados de la etapa previa y se pueden realizar ajustes para la etapa posterior. Se puede decir que estas características son el núcleo del modelo.

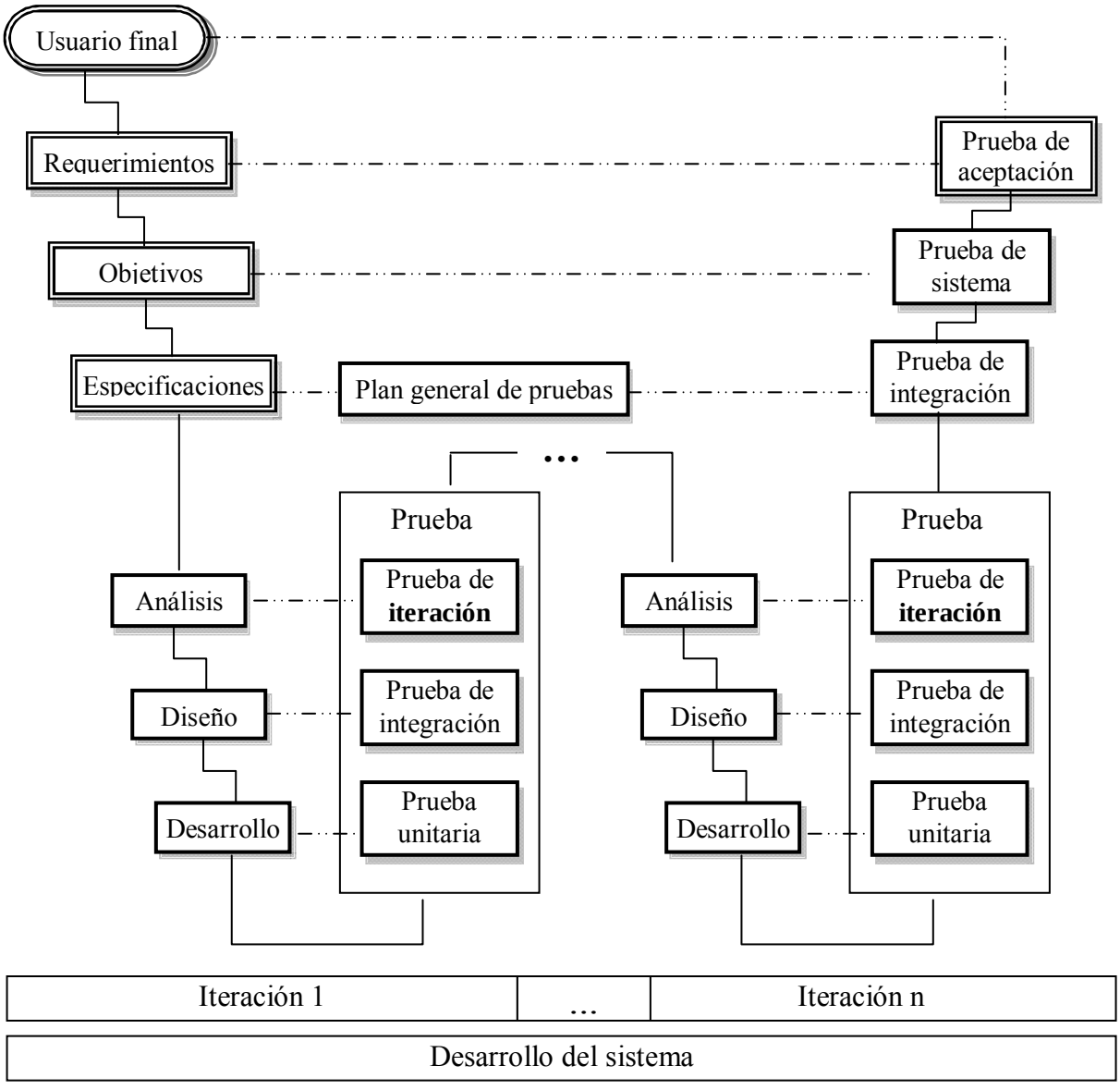

Figura 3.1 Modelo incremental modificado (Modelo W)

Este modelo, propone un mayor número de pruebas al igual que la diversificación de las mismas y con la planeación de estas, se puede saber de antemano cuáles se realizarán y los recursos humanos y materiales que se requerirán. Así que se puede tener una aproximación más precisa en aspectos como: el esfuerzo necesario para realizar la planeación, el personal necesario, la estimación de tiempos para su ejecución, los costos asociados, las condiciones y entornos necesarios. Esta información no se obtiene de forma directa, pero se puede generar a partir del tipo de pruebas a desarrollar y la cantidad.

La posibilidad de disponer de un prototipo funcional en etapas tempranas del desarrollo, con una idea precisa en los requerimientos, tanto por el equipo de desarrollo como del usuario final, puede aminorar el costo y esfuerzo de desarrollo.[27] Si además, se pueden prever las posibles fallas y asegurar la calidad del producto en todo su proceso de desarrollo, hay mayor posibilidad de éxito en el producto final.

Se sacó la etapa de requerimientos de las iteraciones (según el modelo incremental original) y se ubicó al inicio del modelo, ya que esta etapa no puede cambiar de manera drástica durante las iteraciones, ya que el desarrollo se podría extender por periodos incosteables. Además de correr el riesgo de terminar con requerimientos totalmente diferentes a los inicialmente planteados. También disponer de una definición de requerimientos permite planificar las etapas subsecuentes, establecer características del ambiente de desarrollo y la tecnología que se podría utilizar para la resolución del proyecto.

Para mantener la característica de claridad y precisión de requerimientos, en cada iteración se tienen la etapa de análisis, que permite hacer correcciones, ajustes o precisiones en el desarrollo, tomando en cuenta los requerimientos iniciales, con ello se sigue tendiendo la

[27] Vonk, R. (1990). *Prototyping*. Gran Bretaña: Prentice Hall.

flexibilidad del modelo de prototipos para una definición mas precisa de los requerimientos, pero de manera acotada.

Se considera también la definición de objetivos y especificaciones del sistema, en donde se indican los lineamientos generales que el usuario final establece y que debe de cumplir el proyecto al finalizar, estos elementos tienen su contraparte en las pruebas de integración, sistema y aceptación, estas pruebas se realizan una vez terminado el desarrollo de las iteraciones, pero es importante enfatizar que su planeación se debe realizar antes de iniciar las etapas de desarrollo del sistema, estas pruebas son las de mas alto nivel en todo el ciclo de vida.

Se agregó la definición del plan general de prueba, en el nivel de especificaciones, debido a que en esta etapa ya se conocen las características del sistema, con base en esto, se pueden diseñar políticas y lineamientos que utilizarán para la planeación y desarrollo de las pruebas.

La finalidad principal de este modelo es la planeación de pruebas desde el inicio, durante y finalización del proyecto, para asegurar un nivel de calidad en cualquier etapa de desarrollo. Al igual que el cumplimiento de requerimientos sea de calidad en consecuencia, con reducción importante en el número de fallas al finalizar el proyecto y aminorar en gran medida las fallas y tiempos de corrección en las pruebas de integración y de sistema.

En las siguientes secciones del Capítulo se describen cada uno de los elementos de prueba del modelo.

3.1 Plan general de prueba

Se pretende que los requerimientos, objetivos y especificaciones, sean lo mas claros posibles, evitando la ambigüedad, que sean suficientes y los más explicito posible. Para así, poder realizar un diseño y análisis adecuado del desarrollo. También son necesarios para diseñar una planeación

adecuada sobre que tipos de prueba realizar, las estrategias a seguir, el diseño de casos de prueba, métricas a utilizar, etc.

En particular, la definición de requerimientos, permite establecer los parámetros generales de las pruebas en las iteraciones y para la prueba de integración general. Para ello, el plan general de prueba utiliza la documentación de los requerimientos y especificación del usuario final para desarrollar la versión inicial del sistema y las pruebas de aceptación basadas en los requerimientos funcionales, de calidad y de funcionamiento del sistema. Los elementos que se deben generar en esta etapa son:

1. La versión general del plan de prueba. En donde se indique la estrategia a utilizar, las técnicas de prueba, métricas, recursos asignados al proceso de prueba en sus diferentes etapas, los parámetros de calidad de referencia y las políticas a seguir tanto para documentación como para intercambio y análisis de información .
2. Versión general de las pruebas de integración, sistema, y aceptación.

En este plan, también se definen las directrices generales, que se tomarán de base para las especificidades de las pruebas dentro de las iteraciones.

3.2 Pruebas de iteración

Esta prueba, verifica el diseño planteado para una iteración, los puntos a considerar son:

1. Los requerimientos funcionales que abarca la iteración
2. El rendimiento en condiciones límite

Para esta etapa se deben generar los casos de prueba correspondientes a los requerimientos de la iteración, lo cuales deben probar la estructura del sistema, por lo que técnicas de caja negra son adecuadas para tal fin.

Es importante considerar que esta prueba es la última en cada iteración y establecerá las condiciones iniciales para la siguiente iteración, por lo que es importante indicar pruebas de regresión, que permitan en la siguiente iteración validar los resultados debido a las correcciones de errores.

Se debe generar registro de los aspectos del sistema que se han completado y los resultados de las pruebas en función de estos adelantos, con esto se evitan expectativas erróneas tanto del avance como de los resultados de las pruebas. Ya que los resultados de las pruebas pueden diferir cuando se completen otras iteraciones y módulos. Al término de esta prueba, se tiene un indicativo de avance de pruebas respecto al plan general de prueba.

3.3 Prueba de integración de iteración

En esta etapa se realizan pruebas ordenadas, inician en los niveles más bajos hasta llegar a los módulos (o componentes) de más alto nivel. El objetivo principal, es verificar las interfaces y el flujo de datos, para este tipo de prueba se utilizará la técnica top-down. Los elementos a generar en esta etapa son:

1. Los casos de prueba adecuados (que involucren las interfaces modulares)
2. Los elementos necesarios (por ejemplo considerar módulos ficticios) para llevar a cabo las pruebas en función de los módulos completados en la iteración y verificar el flujo correcto de datos entre éstos.

3.4 Prueba unitaria en la iteración

La prueba unitaria en la iteración realiza pruebas sobre uno o más módulos de la iteración, que no se han probado previamente, ya sean porque no se habían terminado o porque se realizaron modificaciones en la iteración actual.

Dado que éste es el nivel más bajo de prueba, se realizan pruebas estructurales y funcionales. La planeación de esta prueba tiene el objetivo de verificar el correcto funcionamiento de la lógica de programación, para ello se requiere del diseño de casos de prueba que revisen los elementos importantes o críticos de la unidad que se analiza su funcionalidad. También se verifica la entrada y salida de datos que procesará la unidad.

Es importante considerar que hay situaciones en las que es innecesario probar ciertas unidades, esos casos son:

- Cuando se tiene una gran cantidad de componentes de software que son reutilizados, las pruebas unitarias pueden omitirse o aplicarse en menor medida.
- Cuando una unidad de software, forma parte de una iteración que se ha revisado varias veces (por ejemplo 12) y dicha unidad no ha sufrido modificación y además ha sido probada en las iteraciones anteriores, sería redundante probarla nuevamente.

Es necesario establecer un criterio para valorar unidades de software para determinar si deben ser probados, dos factores útiles en este caso son[28]:

Factor A: **Nivel de severidad de daño**. La severidad de resultados en el caso de que el módulo o aplicación falle.

[28] Galin, Daniel. *Software Quality Assurance : From Theory to Implementation.* Harlow, Essex ; New York: Pearson/Addison Wesley, 2004.

Factor B: **Nivel de riego en el software**. El nivel de riesgo representa la probabilidad de falla.

Para determinar el nivel de riesgo de un módulo o unidad, se deben examinar los problemas que afectan el riesgo. Estos problemas se pueden clasificar en aplicación/módulo y problemas del programador, algunos elementos son los siguientes:

Problemas aplicación/módulo

- Magnitud
- Complejidad y dificultad
- Porcentaje de software original respecto del software reutilizado

Problemas del programador

- Capacidad profesional
- Experiencia con las características del módulo
- Disponibilidad de soporte profesional (respaldo de conocimiento y experiencia)
- El conocimiento con el programador y la habilidad para evaluar sus capacidades.

3.5 Pruebas de integración general

Finalizado el desarrollo de las iteraciones, se tienen prácticamente los elementos necesarios del sistema que se esta construyendo, esto significa que las unidades de software están completas y disponibles para integrarse e interactuar de forma conjunta.

A pesar de que en cada una de las iteraciones se realizan pruebas de integración, que involucran la interacción de varias unidades de software y que conforme se realizan mas

iteraciones, la interacción puede llegar a ser mayor, no implica que se han probado todas las unidades respecto de las funciones y objetivos que debe cubrir el sistema.

Para ello es necesario diseñar pruebas de integración de unidades, que permitan detectar defectos o errores en las interfaces, ya que una vez ensambladas las unidades de software se puede probar al sistema como un todo y que además sea consistente con los requerimientos. Al igual que en la prueba de integración de iteración, en esta etapa se utiliza una técnica de integración top-down, con ayuda de estrategias de caja negra.

Es importante tomar en cuenta que algunas interfaces e interacción de módulos pueden haberse probado ya sea de forma detallada o parcial en las iteraciones (debido a la importancia o características del sistema), por lo que es recomendable enfocarse en aquellas interacciones que no se han revisado previamente.

En el caso de requerir mayor certeza, tanto en el correcto funcionamiento como en la ausencia de errores, de una o varias interfaces que ya han sido revisadas, es recomendable revisar los casos de prueba y procedimientos aplicados en las iteraciones, para así, diseñar pruebas que revisen aspectos no considerados previamente y con ello asegurar en mayor medida la adecuada funcionalidad de las interfaces y unidades.

También es importante verificar las características de las entradas y salidas en las diferentes interfaces, como lo son el tipo, el orden y la cantidad de datos que se relacionan entre una unidad y otra.

3.6 Pruebas de sistema

La prueba de sistema tiene la finalidad de comparar el sistema respecto de los objetivos inicialmente planteados, para poder realizar esta comparación es necesaria la existencia de elementos que se puedan medir (capacidad de procesamiento, velocidad de ejecución, porcentaje de tolerancia a fallas, etc.) para evitar subjetividad.

Ya que la prueba de sistema está íntimamente relacionada con los objetivos del proyecto, no hay una metodología aplicable, debido a la variedad de condiciones que se pueden plantear en los objetivos. Tampoco es posible utilizar como base, las pruebas de integración, ya que éstas revisan las interfaces y flujo de datos e independientemente de su correcto funcionamiento, no necesariamente cumplen con los objetivos del proyecto.

Para la prueba de sistema, se utiliza la documentación de los objetivos del proyecto y la documentación del usuario. Estos elementos son la base ya que contienen información de lo que el sistema debe hacer y que tan bien lo debe hacer.

Debido a que no hay una metodología general que permita crear algún caso de prueba, esta etapa es una de las más complicadas, ya que evalúa aspectos de funcionalidad y comportamiento, por lo que, una manera indirecta para realizar la prueba de sistema consiste en probar diversos aspectos o categorías del mismo como son:

- Pruebas funcionales
- Pruebas de volumen
- Pruebas de estrés
- Pruebas de uso
- Pruebas de seguridad

- Pruebas de desempeño
- Pruebas de almacenamiento
- Pruebas de configuración
- Pruebas de compatibilidad, configuración, conversión
- Pruebas de instalación
- Pruebas de confiabilidad
- Prueba de recuperación
- Prueba de documentación
- Pruebas de procedimiento

No significa que para cualquier sistema se deban realizar todas las pruebas, dependerá de las características específicas de cada sistema, su aplicación y el entorno de uso. Pero estos aspectos pueden mostrar las discrepancias entre el resultado final y los objetivos del proyecto.

3.7 Pruebas de aceptación

Como se observa en el modelo W figura 3.1, la prueba de aceptación se corresponde en gran medida con los requerimientos del sistema y consiste en comparar éstos con el sistema terminado, además de las necesidades del usuario. En general es una prueba que realiza el usuario final en un entorno (sistema operativo, hardware, etc.) similar al de instalación del sistema o inclusive en el equipo destinado para su uso, también es operado en lo posible por los diversos niveles de usuarios finales, con el propósito de evaluar su funcionamiento en un entorno de uso real.

Es importante acordar desde un inicio los lineamientos que se aplicarán en las pruebas de aceptación (en general establecidos en el contrato), por lo que conjuntamente con la definición de

requerimientos se definen criterios de aceptación, se requiere de la participación del usuario final junto con el equipo de pruebas, tanto para el diseño, como para la ejecución de esta última etapa.

Los objetivos principales de esta etapa son: probar que el sistema está listo para su operación y realizar pruebas que se han planificado con el propósito de establecer la confiabilidad y facilidad del uso del sistema por parte del usuario. Esto último para no caer en la realización de pruebas exhaustivas e improvisadas para encontrar alguna falla en el sistema por parte del usuario.

Se pueden aprovechar algunas pruebas ejecutadas en la "prueba de sistema", también es posible realizar simulaciones de operación real en un periodo de tiempo (por ejemplo un día de trabajo), con información real y evaluar los resultados.

En esta etapa también se validan los procedimientos de uso, las restricciones, condiciones de operación y la documentación

3.8 Uso de métricas

Una parte importante del modelo, es la parte iterativa. Es necesario disponer de parámetros, que muestren determinados aspectos, ya sea de calidad, de eficiencia, económicos, etcétera, para tener datos que permitan observar el comportamiento en el transcurso de las iteraciones.

Las métricas son una herramienta que nos permite obtener información acerca de aquellos parámetros que nos interesen, para lo cual se utiliza el marco de calidad de software abordado en la sección 2.7. La ventaja de utilizar este marco de referencia es que permite evaluar los elementos que no cumplen con criterios de calidad y conllevan a propiedades de calidad inadecuada o defectuosa. Con lo cual se puede realizar la detección de errores.

3.9 Consideraciones finales

En el Modelo W, se pueden apreciar dos tipos de pruebas, una de alto nivel que abarca las pruebas de integración, sistema y aceptación, y otra de bajo nivel que corresponde a las pruebas específicas a cada iteración.

En ambos tipos, el objetivo del modelo, consiste en realizar una detección temprana de errores en cualquier nivel del desarrollo, con un mayor énfasis en la etapa iterativa. Para lograr esto, se requiere de la planeación y diseño de pruebas, junto con la planeación de las diferentes etapas de desarrollo y al final de cada una, comprobar las metas fijadas con los resultados de las pruebas, y así maximizar la búsqueda de errores.

Diseñar pruebas a la par del desarrollo del sistema, permite tener una mejor perspectiva para determinar qué elementos probar, de qué manera probarlos y definir cuáles son los resultados aceptables para conseguir mayor calidad del sistema.

Para seleccionar un tipo de métrica o algún marco de referencia para la evaluación de software, existen varias alternativas para diversos aspectos de calidad, productividad, seguridad, claridad, eficiencia, etc., que se pueden aplicar en los niveles que se plantean en el Modelo W, sin embargo, este aspecto puede ser inclusive, un tema para otro trabajo de tesis.

El alcance de este trabajo, no considera a profundidad el aspecto de las métricas y sólo se limita a utilizar un marco de trabajo, el cual se puede aplicar a gran diversidad de desarrollos de software.

Capítulo 4

4 Sistema para la planeación de pruebas

En este capítulo se presenta un prototipo de software que apoya en la planeación y el control de las pruebas propuestas en el Modelo W.

4.1 Análisis

4.1.1 Características del sistema

El modelo W propone una serie de pruebas, con diferentes características a realizarse en determinadas etapas del desarrollo. Se debe obtener la información que requiere cada tipo de prueba y asociarse a la etapa e iteración en que se aplica. Una vez que se ejecute alguna prueba, también se deben registrar los resultados, si fue exitosa o no, si se encontraron errores y cuantos, alguna observación acerca de la ejecución o si no fue posible completarla.

Está entre las actividades mencionadas, el análisis de resultados, el cual permite conocer las características del desarrollo, su calidad o deficiencias, entre otras cosas. Este proceso puede considerarse como el más importante pero es también el más complicado, ya que depende de forma directa de las características del sistema, su propósito, el entorno en el que se va a aplicar, el número de usuarios, etc. Por tanto la forma de realizar el análisis puede tener diversas alternativas.

Dado que la cantidad de pruebas a realizar y la información generada en este proceso, pueden llegar a ser importantes en volumen e involucra a varias personas, se requiere de una

herramienta de software que permita controlar la planeación de pruebas y el registro de resultados.

También es importante llevar un control del personal que realiza cada una de las operaciones y el momento en que las realiza, para lo cual se requiere establecer roles.

4.1.2 Identificación de requerimientos

En el proceso de prueba existen dos tipos, de alto y bajo nivel. Cada tipo se enfoca a aspectos particulares del software. El sistema a desarrollar, debe mostrar las particularidades de cada prueba y proveer los mecanismos necesarios para registrar los datos necesarios para cada tipo, por lo que es importante precisar la información que debe considerar cada prueba e identificar la etapa a la que pertenece.

El sistema debe almacenar y permitir la consulta del plan general de pruebas, el cual contendrá la información de los aspectos globales en lo concerniente a las pruebas, objetivo, descripción, políticas, tipos de pruebas, estrategias y métricas a utilizar en todo el proyecto.

También se requiere registrar los datos del proyecto, como el nombre, tiempo estimado de duración, una descripción general, estimación del número de iteraciones, fecha de inicio del proyecto, el personal disponible para el desarrollo, sus datos, al igual que la especialidad de cada miembro del equipo y el rol que desempeñará.

Se debe llevar registro del inicio y finalización de cada tipo de pruebas, recabar la información del estatus (si está en ejecución, si concluyó, no se pudo realizar, etc.), los criterios de aceptación de la misma, quién realiza la planeación, ejecución y revisión de la prueba.

En el caso de las pruebas de iteración, se debe controlar además del tipo de prueba, la iteración a la cual corresponde. En todos los casos registrar si es el caso, errores, fallas y defectos,

u observaciones acerca de cualquier situación anómala que se haya presentado. En todo momento se debe disponer de algún mecanismo que permita revisar tanto la planeación de pruebas, como el estado de las mismas, de los distintos tipos de pruebas y de cada una de las iteraciones.

4.1.3 Descripción de la solución

La alternativa de solución para este problema consiste en englobar todas las pruebas como un solo proceso y realizarlo en tres etapas: planeación, ejecución y resumen. Este proceso se muestra en la figura 4.1.

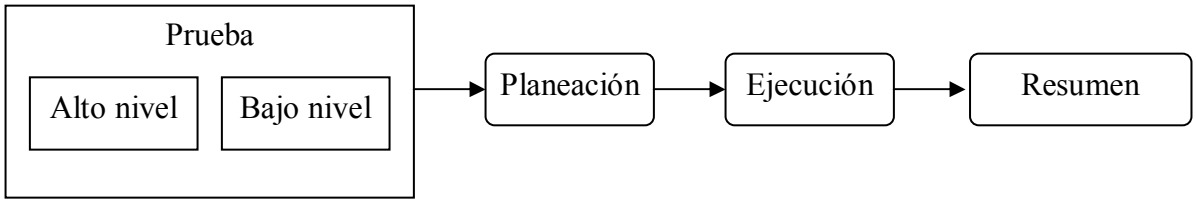

Figura 4.1 Esquema general del prototipo

Desde esta perspectiva, se desarrolla el sistema de forma que la información de cada prueba, se almacena en tres etapas: la planeación registra los datos particulares de cada prueba y quién la realizó; la ejecución, almacena los datos de cuándo se ejecuta la prueba, quién la realiza y características generales; finalmente en el resumen de la prueba se registra cuándo finalizó la ejecución, las incidencias encontradas, quién revisa los resultados, si se concluyó la prueba adecuadamente y si se cumplen los criterios de aceptación.

Con los datos obtenidos se generan reportes por tipo de prueba, por fechas, por estatus de las pruebas o por usuarios. También se lleva registro del personal involucrado en el sistema y las actividades a las que está asignado, con la finalidad de realizar seguimiento de quien realiza las actividades en todo el proceso de desarrollo.

En la figura 4.2 se muestra el diagrama de actividades, que modela la solución para el problema planteado, también se toma como base la perspectiva de la figura 4.1

4.1.4 Resultados esperados

Las metas que se pretende conseguir, se listan a continuación:

- Controlar, a través del sistema el manejo de información, de los diversos tipos de pruebas y facilitar el registro de la información (uso de plantillas de captura).
- El mecanismo de registro de información, de los procesos de prueba debe ser incremental.
- Facilitar el seguimiento de las pruebas realizadas a lo largo del desarrollo.
- Proveer de control de pruebas que se realizan en las iteraciones.
- Posibilitar listados de las pruebas conforme a criterios como: fechas, estatus de la prueba, persona que ejecuta la prueba, etc.
- Mantener un control de las actividades y del personal que opera el sistema.

4.1.5 Limitantes

En cuanto a las características requeridas del sistema planteado, existen procesos que pueden llegar a un alto nivel de automatización, como es el caso del análisis de resultados, generación de estadísticas, la aplicación e inclusión de métricas de acuerdo al tipo de proyecto, el conteo de líneas de código del sistema por cada iteración, etc.

Sin embargo, debido a la complejidad para realizar estas tareas y a que el alcance de este trabajo sólo consiste en desarrollar un prototipo, únicamente se considera el registro de la información en las etapas que se muestran en la figura 4.1 y el resultado de incidencias, que pueden obtenerse con ayuda de otras herramientas de software (por ejemplo el conteo de las líneas de código).

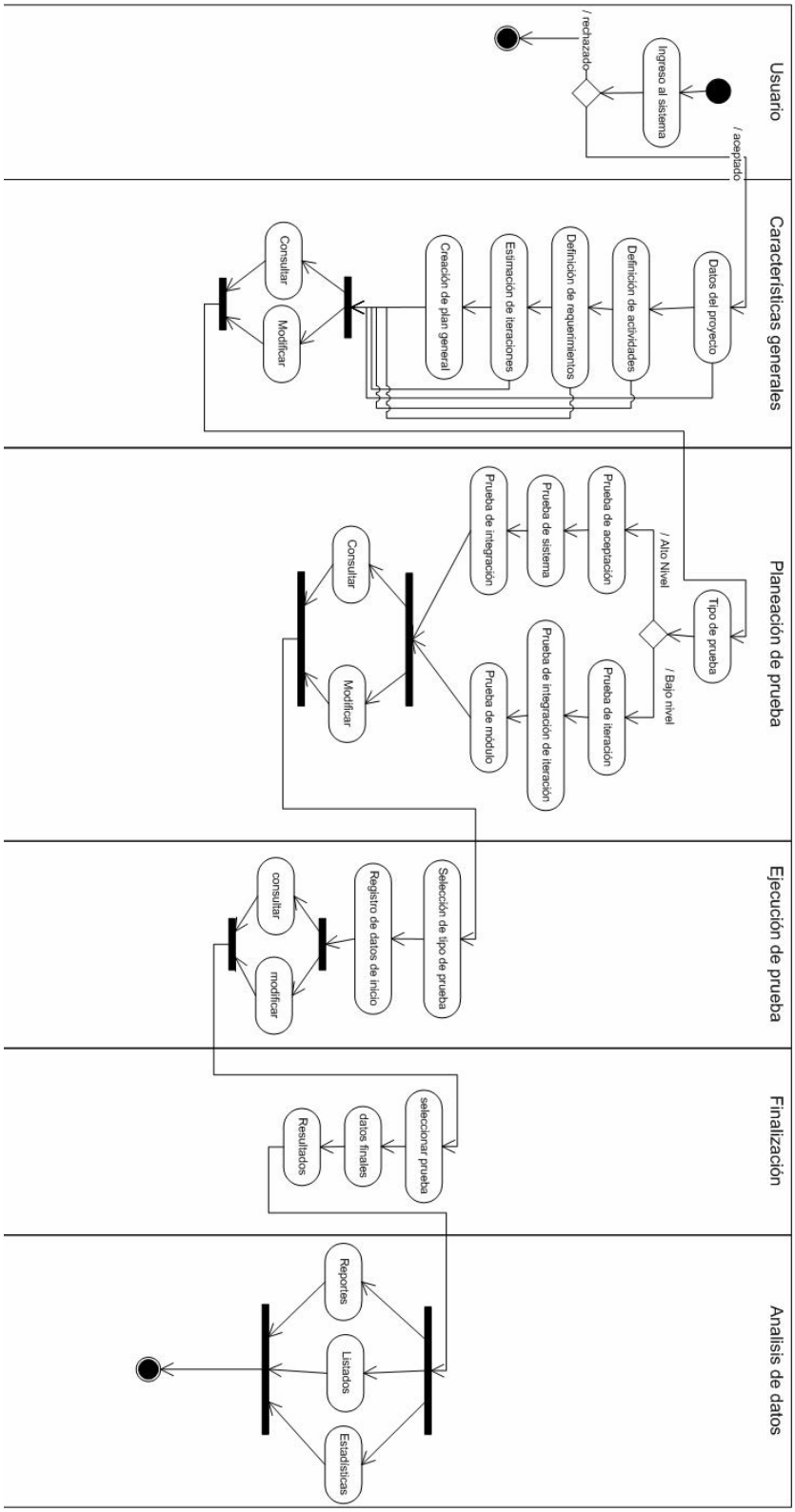

Figura 4.2 Diagrama de actividades del prototipo

4.2 Diseño

4.2.1 Elementos usados en el diseño

El prototipo desarrollado se construyó utilizando Delphi versión 7.0, ya que permite un desarrollo de interfaces relativamente rápido, es un leguaje con manejo orientado a objetos y a eventos con la cual se desarrolló la interfaz y el manejo de la lógica de negocios.

Para el almacenamiento de la información se utiliza el motor de base de datos MySQL, que a pesar de no contar con todas las características de un manejador de base de datos, es muy flexible en varias plataformas de cómputo, se encuentra bajo el esquema de software libre y realiza un manejo multiusuario.

Finalmente la comunicación entre la aplicación y el motor de base de datos se realizará por medio de ODBC (*Open Database Connectivity*). Lo cual permitirá flexibilidad de comunicación entre plataformas.

El esquema con la integración de estos elementos se muestra en la figura 4.3, como se observa, el desarrollo del prototipo está implementado en plataforma Windows.

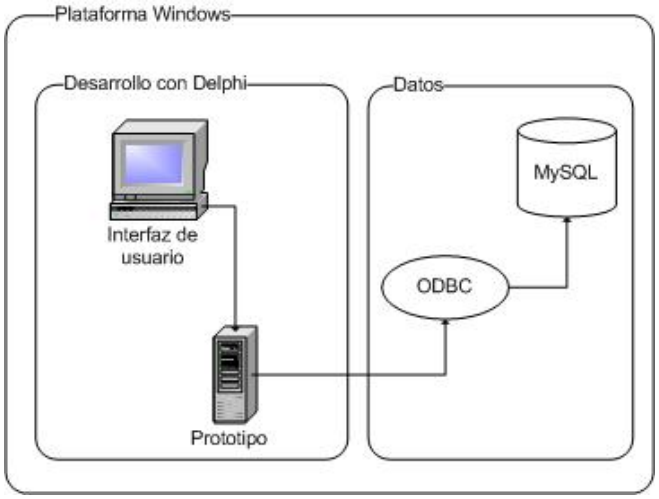

Figura 4.3 Estructura de funcionamiento del prototipo

4.2.2 Definición de requerimiento de datos

Tomando en cuenta las características mostradas en las figuras 4.1 y 4.2, se plantea una estructura de almacenamiento figura 4.4, que además de cubrir dichas características, contempla los requerimientos del proyecto y es flexible porque puede almacenar la información de más de un proyecto. Esta estructura considera cierta independencia de datos que puedan utilizarse para un desarrollo futuro, con más sofisticación.

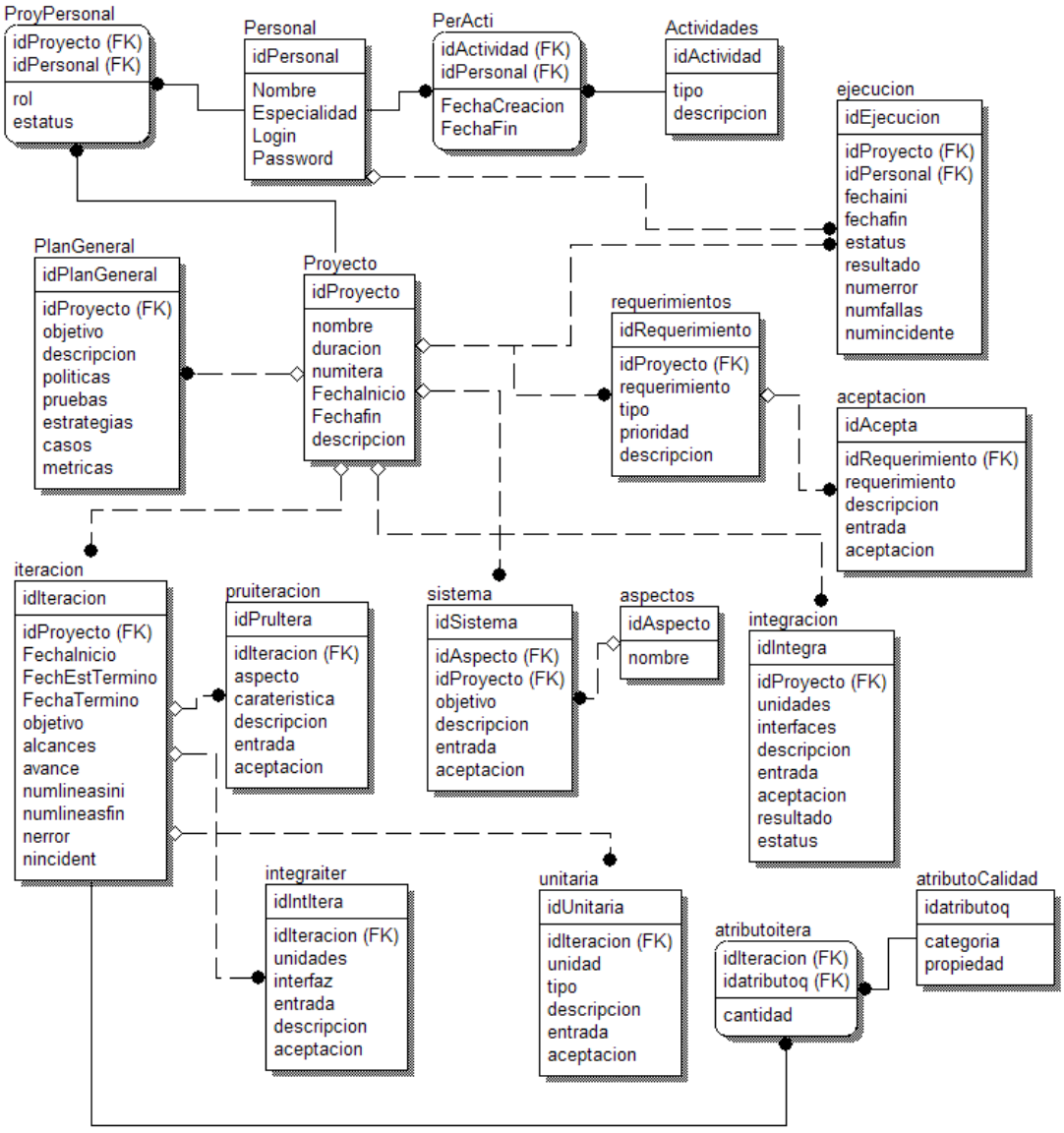

Figura 4.4 Esquema de la base de datos del prototipo

4.3 Construcción, interfaces y resultados

El desarrollo del sistema consta de cinco módulos principales que son:

- Manejo de los datos generales del proyecto.
- Manejo de la información de las iteraciones.
- Administración de usuarios del sistema.
- Planificación y resumen de los diferentes tipos de prueba.
- Despliegue de la información por tipos de pruebas.

La interfaz principal del sistema consta de siete menús, que a continuación se enuncian y se da una breve explicación:

Archivo. Permite la conexión y desconexión al sistema, por parte del usuario.

Proyecto. Contiene las opciones que permite ingresar la información general del proyecto

Planeación. Permite ingresar la planeación de todas las pruebas de alto y bajo nivel, además de poder realizar la consulta de las mismas.

Pruebas. En este rubro se ingresa la información de los procesos de ejecución y finalización de todas las pruebas.

Reportes. Generación de reportes organizados por tipo de prueba.

Personal. Manejo del acceso y datos de los usuarios del sistema, captura y asignación de actividades.

Ayuda. Información de apoyo para el uso del sistema.

En las siguientes figuras 4.5 a 4.9 se muestran las interfaces del plan general de prueba, un ejemplo de la planeación de prueba, del registro de su ejecución, su finalización y la captura de defectos localizados al final de la iteración.

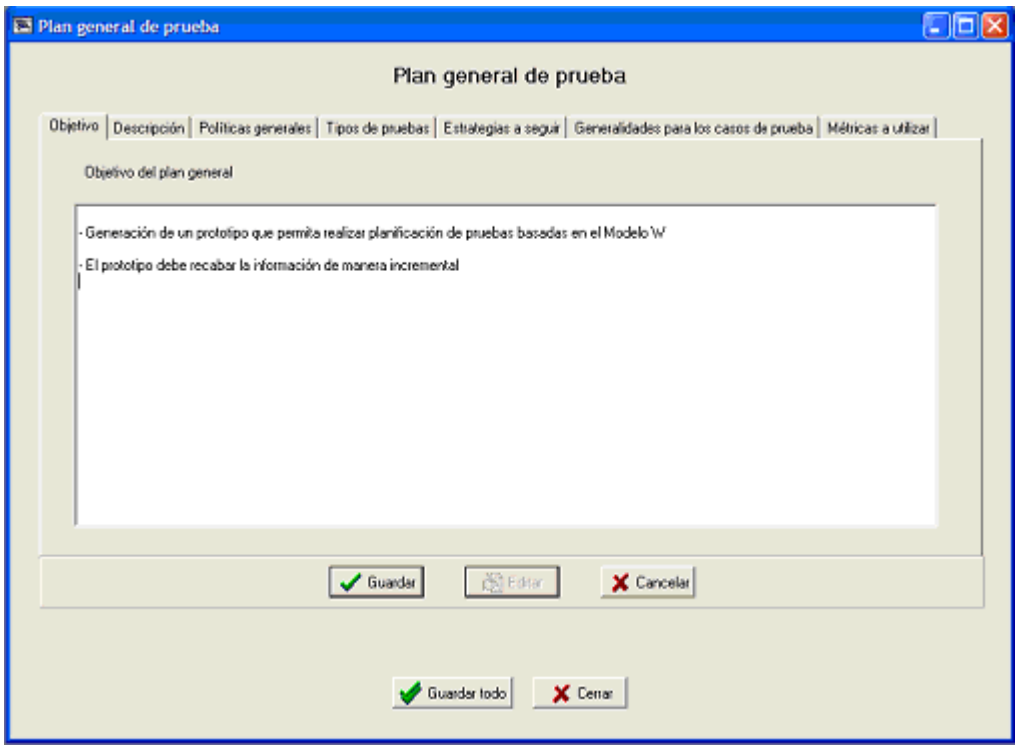

Figura 4.5 Interfaz del plan general

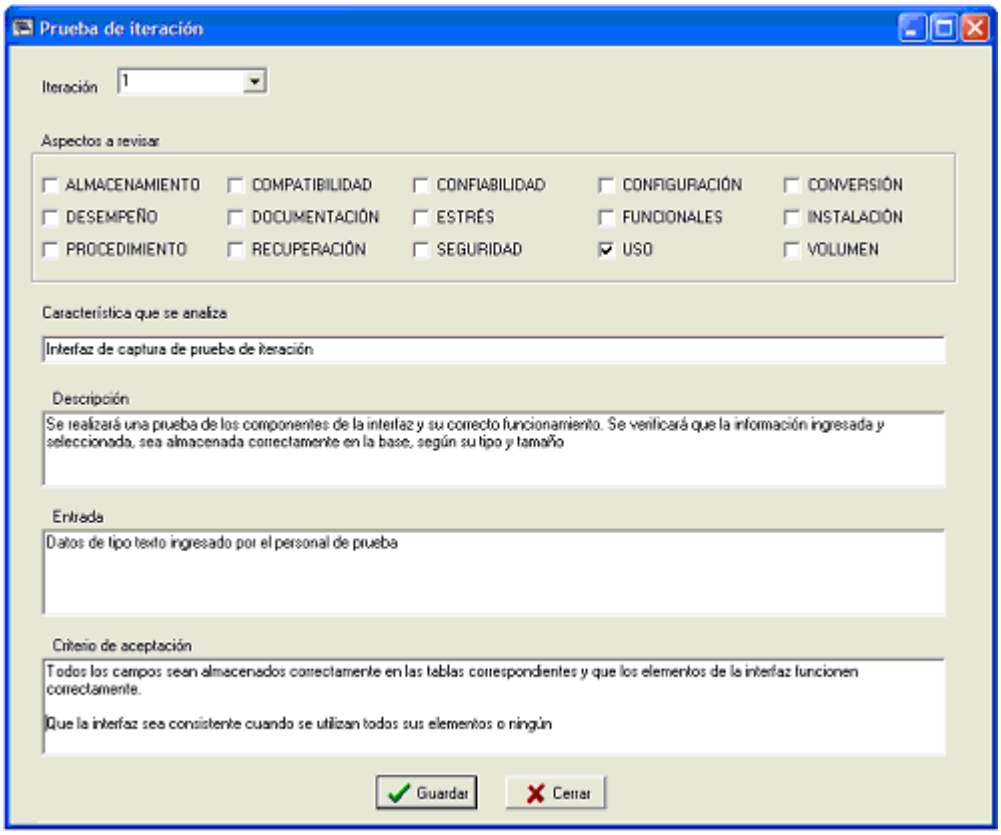

Figura 4.6 Ejemplo de captura de prueba de iteración

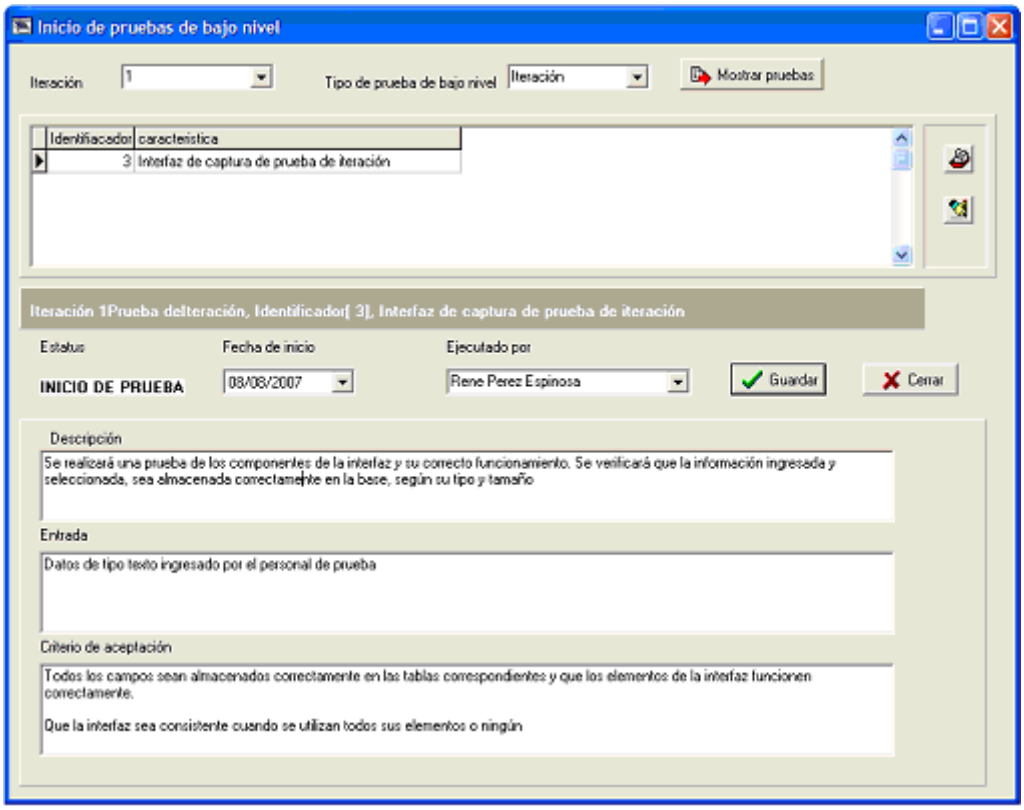

Figura 4.7 Inicio de ejecución de pruebas de bajo nivel

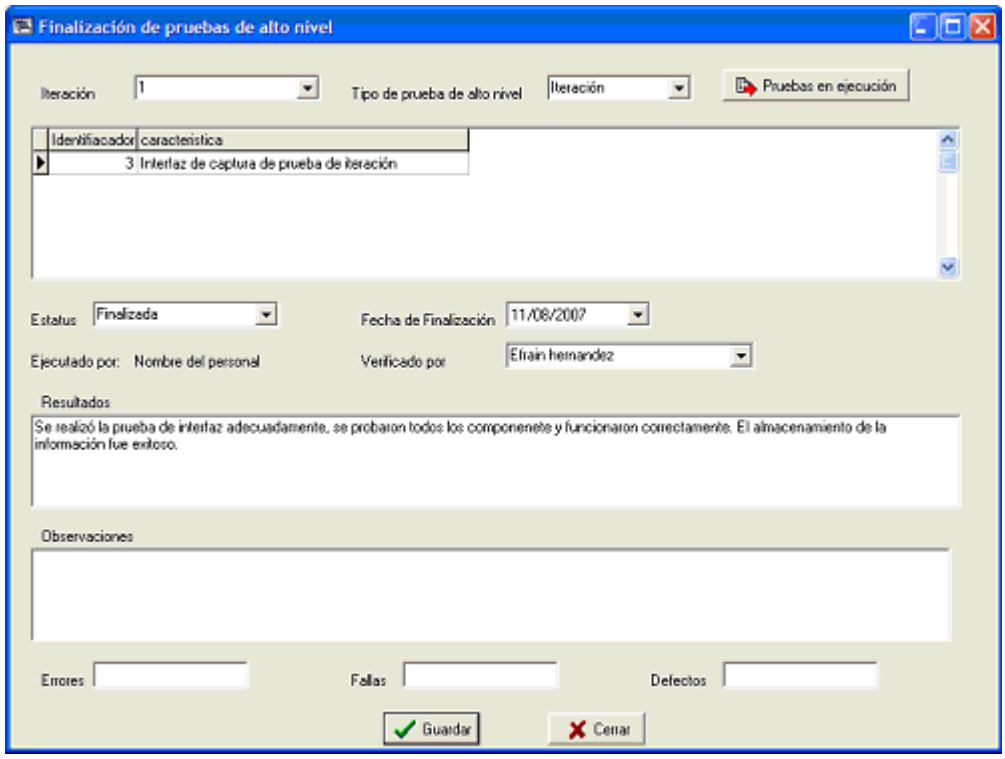

Figura 4.8 Ejemplo de finalización de pruebas de bajo nivel

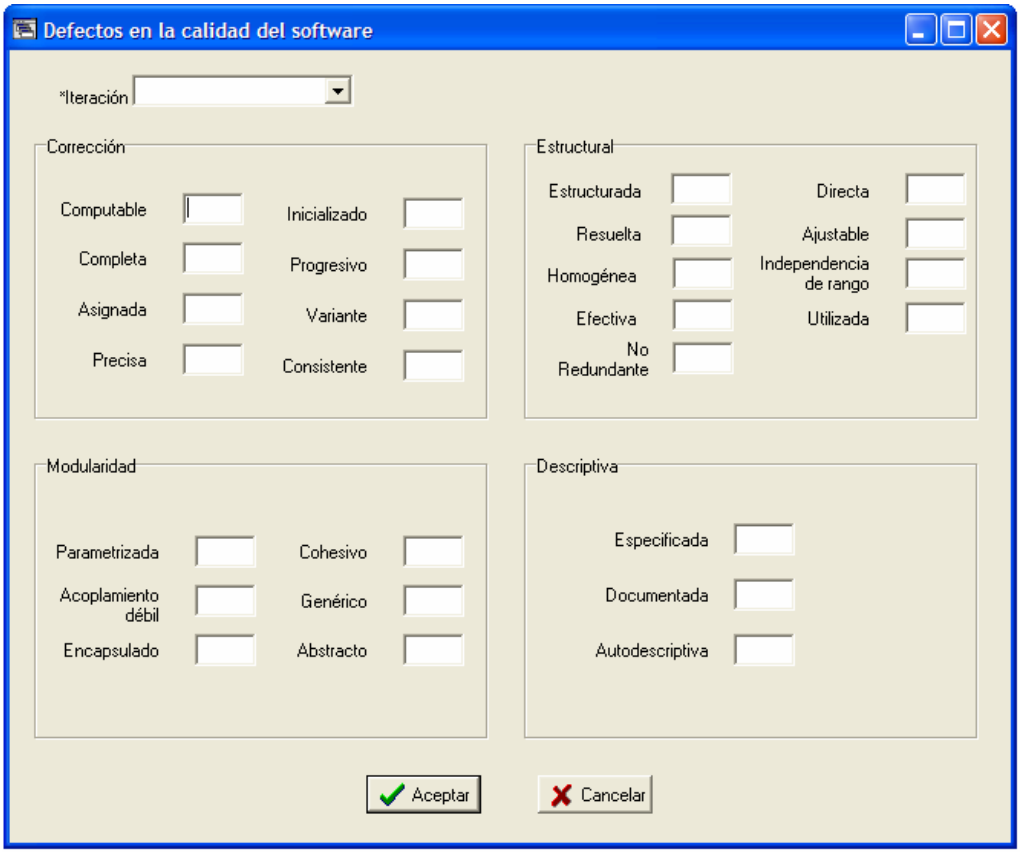

Figura 4.9 Evaluación de defectos localizados al final en iteración

4.4 Consideraciones finales

El prototipo desarrollado, permite realizar las operaciones básicas para realizar la planificación de las pruebas a lo largo de las etapas de desarrollo de software, de la forma en que las contempla el Modelo W.

Dado que el alcance principal de esta tesis no es la de desarrollar una aplicación, el software generado, permite por tanto observar una serie de requerimientos deseables, que se pueden tomar como base para realizar una mejora en la definición de requerimientos, para poder desarrollar una aplicación de otro tipo, que disponga de características que ayuden en mayor medida a la actividad de pruebas utilizando el Modelo W.

Finalmente se pueden sugerir mejoras a este prototipo, se le pueden agregar otras funcionalidades como lo son: herramienta para el conteo de líneas de código, que considere las características sintácticas de diversos lenguajes de programación o que se puedan definir dichas características. La posibilidad de definir métricas utilizando la información recabada. La posibilidad de generar reportes definidos por el usuario. Mecanismos para realizar análisis de avance, tiempo empleado, esfuerzo, estadísticas, estimaciones etc. Todas estas características se plantean como trabajo futuro.

Capítulo 5

5 Pruebas de prototipo

En este capítulo se hace el análisis de un sistema real, cuya construcción se basó en el modelo de prototipo incremental. En primer lugar se hace una descripción general del sistema y las iteraciones que se realizaron, posteriormente se comparan los resultados de las pruebas realizadas con los resultado de aplicar el modelo generado en este trabajo, para poder evaluar si la aplicación del modelo puede incrementar la calidad de en un desarrollo de software.

Ya que el sistema está en operación y que las características de desarrollo y en específico las pruebas realizadas se basaron en otros criterios, que no se ajustan completamente al modelo propuesto, se decidió aplicar el Modelo W con respecto a las pruebas unitarias.

Por tanto se realizó la planeación y aplicación de las pruebas unitarias basándose en el marco de calidad de software de Dromey propuesto en el subtema 2.7. Se revisó el código final de cada iteración, se clasificaron los defectos encontrados y se realizó un análisis de los posibles problemas de calidad en el software, para comparar los resultados respecto de las pruebas realizadas sin la aplicación del Modelo W.

5.1 Descripción de la aplicación a probar

La aplicación "Espacio de Aprendizaje"[29] (EA) es una herramienta de apoyo para impartir clases de la Maestría en Bibliotecología y Estudios de la información que se imparte en la Facultad de

[29] http://infocuib.laborales.unam.mx

Filosofía y Letras y es corresponsable el Centro Universitario de Investigaciones Bibliotecológicas (CUIB), en la UNAM.

El EA se desarrolló en el CUIB debido a la necesidad de impartir clases de maestría en otras sedes del país, como Yucatán y San Luís Potosí. También se dan asesorías institucionales y cursos de actualización en entidades fuera del Distrito Federal, por parte de los investigadores del CUIB.

Para atender esta demanda, se planteó la necesidad de desarrollar un sistema con interfaz Web, que permitiera la interacción entre alumnos (en general) y tutor (investigadores) de manera que se redujera el número de veces que un profesor se trasladara a la sede foránea y así cubrir una mayor demanda de clases, minimizar los gastos de transporte y hospedaje, y tener un contacto más periódico en cuanto al avance académico.

Dado que las clases ya se estaban impartiendo y no se tenían suficientes recursos humanos y de cómputo, se decidió trabajar bajo el modelo de desarrollo de prototipo, para aprovechar los recursos disponibles y definir con mayor precisión los requerimientos. Bajo estas condiciones, los aspectos que debían considerarse en la aplicación eran la portabilidad, su funcionamiento en LINUX pero escalable a sistemas UNIX e inclusive la posibilidad de usarse en Windows. Para el desarrollo se utilizó el lenguaje de programación PHP y el motor de base de datos MySQL.

5.1.1 Características del prototipo

El objetivo principal de la aplicación es el apoyo para impartir clases o cursos de actualización a distancia a través de Internet, que disponga de mecanismos de comunicación asíncrona y sobre todo, que sea de fácil uso y no requiera gran inversión, ni especialización para configurar un curso.

La aplicación maneja elementos públicos (al que pueden acceder todos los participantes del curso) y privados (información correspondiente para cada participante). Dentro de los elementos públicos, se encuentran aquellos que describen la estructura del curso, objetivos, la forma de trabajo, referencias impresas o electrónicas que se utilicen o apoyen al desarrollo del curso, información del profesor y participantes, foros de discusión y el calendario de las actividades a realizar.

Los elementos privados implican el trabajo desarrollado por cada alumno, su evaluación, comentarios a sus actividades, retroalimentación por parte del profesor, control de acceso al sistema y la información que genere y agregue a la aplicación como parte de sus actividades.

Otra característica es la de crear un curso y agregar los contenidos, por medio de la misma interfaz. Para ello se cuenta con un módulo administrativo, que permite configurar la información general del curso, su estructura, el ingreso de los participantes, control sobre las actividades a realizar, los periodos en que se llevarán a cabo y la manera en la que se realizarán.

El Espacio de Aprendizaje ha evolucionado a través de seis incrementos en el prototipo y su versión final actualmente se utiliza para impartir cursos tanto para nivel de maestría, como de licenciatura, cursos de educación continua y para diplomados. Se ejecuta en equipos SUN con sistema operativo Solaris y da soporte a 15 materias de maestría, con un promedio de 12 alumnos por materia, en promedio se utilizar para impartir 5 cursos de educación continua al año y al menos un diplomado.

5.1.2 descripción de las iteraciones

En seguida se describen los objetivos que fueron cubiertos en cada iteración y que también describen las características que se fueron agregando a lo largo del desarrollo de la aplicación.

Iteración 1

- Elaboración de la interfaz general con los rubros principales de la aplicación: presentación del curso, objetivos, temario, actividades, foros, bibliografía, recursos de aprendizaje, archivos, evaluación, alumnos y profesor, y
- Programación de las opciones del foro de discusión, captura y listado de los participantes del curso.

Iteración 2

- Generación de la estructura de la base de datos,
- Generación de la interfaz de acceso con login y password,
- Registro y validación de acceso a la aplicación, y
- Programación de las opciones relacionadas con los alumnos: alta, modificación, eliminación y procedimientos para calificar actividades.

Iteración 3

- Mejoras en la interfaz gráfica,
- Módulo para enviar y descargar archivos, y
- Calendario de actividades del curso.

Iteración 4

- Procedimientos para crear y manipular evaluaciones,
- Mecanismos para registrar comentarios y retroalimentación de las actividades de los alumnos, y
- Registro de bitácoras de acceso, por IP, fecha y hora de entrada y cierre de sesión.

Iteración 5

- Módulo para la configuración de los rubros generales del curso,

- Módulo para la creación y manipulación de actividades en el curso,
- Mecanismos para enviar correos entre los alumnos desde la aplicación,
- Generación de carpetas de alumnos, conteniendo todos los archivos propios del alumno, la calendarización de actividades a realizar en el desarrollo del curso y su respectiva calificación de las actividades concluidas, y
- Listados por diferentes criterios, para manejo de actividades de alumnos, para visualización del profesor.

Iteración 6

- Generación del módulo para ingreso de temario (temas y subtemas), con control de fechas y generación de detalle de cada tema con archivos asociados, y
- Módulo para la generación de cuestionarios de evaluación.

Iteración 7

- Integración de editor de HTML para la elaboración del curso,
- Mejoras en el registro de sesiones,
- Mejoras en el despliegue del foro de discusión y
- Creación de aplicación de mensajes instantáneos.

5.2 Resultados de las pruebas realizadas

En el desarrollo del EA se hizo mayor énfasis en las pruebas de aceptación. Dichas pruebas se realizaron en conjunto con los usuarios del sistema (profesores, alumnos) y con los operadores involucrados en el desarrollo de un curso (personal que introduce contenidos, que realiza proceso de captura de datos de alumnos y profesores, etc.).

Se llevaron a cabo pruebas de desempeño, de usabilidad, funcionalidad, etcétera, sin embargo no se realizó una documentación adecuada. Tampoco se documentaron las pruebas unitarias, por lo que la dificultad en llevar un control de los errores corregidos, las pruebas aplicadas y los módulos probados resultó un tanto difícil.

El comportamiento en cuanto a líneas de código fuente por iteración se muestra en la figura 5.1 y la tabla. El conteo excluyó líneas en blanco y los comentarios se contabilizaron por separado, también se hace una diferenciación entre las líneas de código en PHP y la codificación en HTML, aunque en los cálculos se toma el total de líneas ya que la revisión implica ambos.

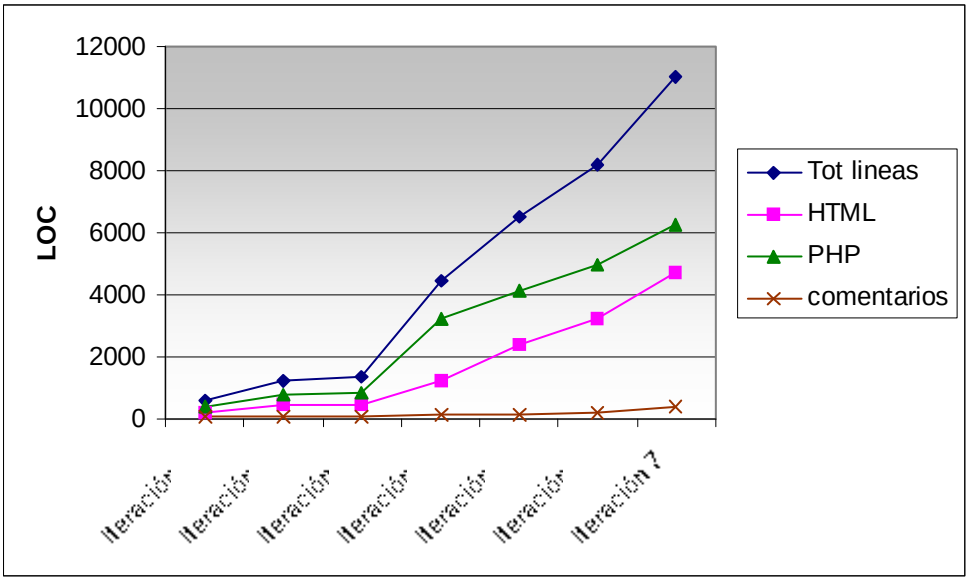

Figura 5.1 Grafica que muestra las líneas de código por iteración

Iteración	HTML	PHP	Comentarios	LOC
1	175	378	39	553
2	468	779	53	1247
3	483	848	40	1331
4	1250	3196	134	4446
5	2369	4147	151	6516
6	3200	4984	179	8184
7	4741	6280	366	11021

Tabla 5.1 Datos de líneas de código por iteración

Los datos que se recabaron al finalizar las iteraciones corresponden a las pruebas unitarias y consideraron los aspectos de la tabla 5.1.

Aspecto revisado	Tipos de errores
Detección de errores en la programación	Errores de sintaxis Errores de variables Errores en resultados Inconsistencia de datos
Probar valores	Errores de valores límite Errores por valores fijos
Prueba en la interfaz	Errores de interfaz Errores de navegación Archivos innecesarios o duplicados
Prueba de compatibilidad	Errores de compatibilidad con navegadores Errores de compatibilidad con sistema operativo

Tabla 5.2 Aspectos revisados en las iteraciones

Los resultados de las tablas 5.2 a la 5.5, muestran los errores encontrados al final de cada iteración y es importante mencionar que esta información contabiliza errores que se encontraron después de haber realizado correcciones, estos errores se identificaron por el uso del prototipo o en la instalación en un entorno real.

En la tabla 5.2 se muestran los errores cuyo origen está relacionado con errores en la programación, ya sea por los usos inadecuados de la sintaxis, variables mal asignadas o con errores de tipo, operaciones con resultados erróneos o inconsistentes.

Iteración	Errores de variables	Error de sintaxis	Error en resultados	Inconsistencia de datos
Iteración 1	1	1	0	0
Iteración 2	0	0	0	0
Iteración 3	0	0	1	1
Iteración 4	0	0	1	1
Iteración 5	0	0	0	0
Iteración 6	0	0	1	1
Iteración 7	0	1	3	1

Tabla 5.3 Detección de errores en la programación por iteración

La tabla 5.3 muestra los errores obtenidos al hacer pruebas con datos de entrada y probar los valores máximo y mínimo, longitudes de los campos, tipos de datos.

Iteración	Valores límite	Valores fijos
Iteración 1	0	0
Iteración 2	2	1
Iteración 3	4	4
Iteración 4	3	1
Iteración 5	12	0
Iteración 6	8	0
Iteración 7	8	0

Tabla 5.4 Resultados de pruebas de valores por iteración

En la tabla 5.4 se registran los errores relacionados con el funcionamiento de la interfaz, su navegación y la duplicidad de archivo, archivos obsoletos que no han sido eliminados del proyecto o no corresponden a la versión de la iteración.

Iteración	Error de interfaz	Error de navegación	Archivos innecesarios
Iteración 1	1	0	1
Iteración 2	7	0	0
Iteración 3	3	3	11
Iteración 4	1	0	3
Iteración 5	5	1	0
Iteración 6	1	3	6
Iteración 7	2	2	5

Tabla 5.5 Resultados de pruebas de interfaz por iteración

La tabla 5.5 muestra los errores que se relacionan con el sistema operativo en el que se está probando la aplicación o con el entorno de operación (versión de servidor Web, de PHP, configuraciones, etc.).

Iteración	Error de compatibilidad	Error en SO
Iteración 1	0	0
Iteración 2	0	0
Iteración 3	1	1
Iteración 4	1	1
Iteración 5	1	0
Iteración 6	0	0
Iteración 7	0	0

Tabla 5.6 Resultados de pruebas de compatibilidad por iteración

La gráfica de la figura 5.2 muestra la frecuencia de errores detectados por categoría, de donde se puede observar que la mayor concentración de errores son los relacionados con los valores y con la interfaz

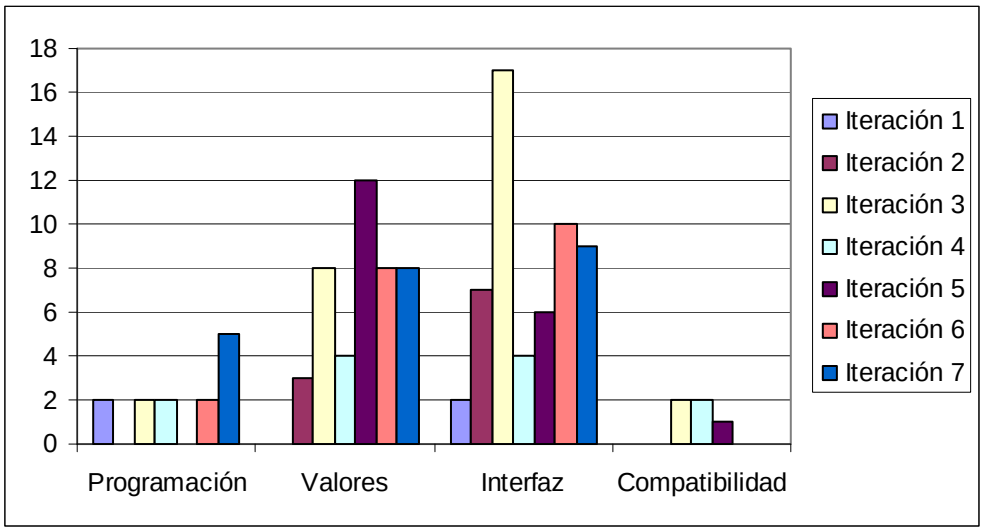

Figura 5.2 Gráfica de resultados de tipo de prueba por iteración

La distribución total de errores encontrados por cada iteración se muestra en la figura 5.3, realizando el cálculo de densidad de errores (DE) expresado en porcentaje con la siguiente formula:

$$DE = \left(\frac{\text{Número de errores}}{\text{LOC}} \right) 100\%$$

Tomando en cuenta los resultados de la tabla 5.6, se tiene que el promedio de errores por iteración es de 16.5 y el promedio de densidad de error es de 0.67%

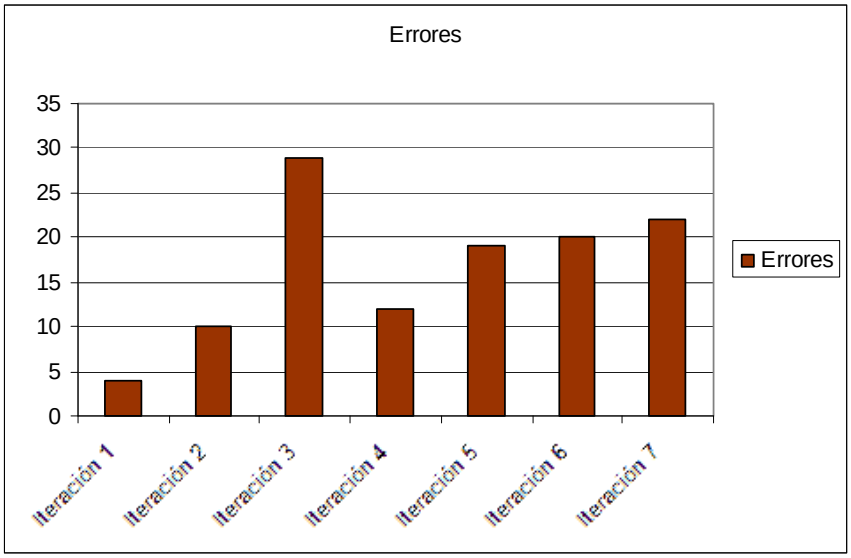

Figura 5.3 Errores detectados por iteración

Iteración	Total de errores	LOC	%DE
1	4	553	0.72%
2	10	1247	0.80%
3	29	1331	2.18%
4	12	4446	0.27%
5	19	6516	0.29%
6	20	8184	0.24%
7	22	11021	0.20%

Tabla 5.7 Porcentaje de densidad error por iteración

5.3 Resultados con el modelo generado

Puesto que la aplicación ya esta terminada, se decidió aplicar pruebas a nivel de iteración y en específico, pruebas unitarias por dos razones:

La primera, debido a que es el nivel más bajo de prueba y hasta cierto punto se pueden excluir ciertas características de diseño y análisis.

La segunda, debido a que se puede observar el impacto de realizar una planeación y aplicación de pruebas en iteraciones, y así comparar los resultados que se obtuvieron en el desarrollo de la aplicación.

Tomando como referencia el modelo de calidad de software presentado en la sección 2.7, se emplearon los criterios de dicho modelo en la aplicación (ver apéndice A). Se presentan los datos obtenidos de la revisión y clasificación de defectos, en las tablas 5.7 a 5.10.

En la tabla 5.7 se muestran los defectos relacionados con variable y que pueden llegar a presentar problemas bajo ciertas circunstancias.

Iteración	Completa	Asignada	Precisa	Inicializado	Consistente
I1	0	1	0	1	2
I2	2	0	0	1	0
I3	0	0	0	0	0
I4	3	0	0	1	0
I5	135	0	0	2	0
I6	141	0	6	0	0
I7	89	1	0	0	0

Tabla 5.8 Resultado de las propiedades de corrección por iteración

La tabla 5.8 presenta los defectos estructurales que pueden llegar a presentar los bloques de código y las relaciones entre estos.

Iteración	Estructurada	Homogénea	Efectiva	No redundante	Directa
I1	0	0	8	0	0
I2	12	0	17	0	0
I3	7	0	19	0	0
I4	6	0	8	0	2
I5	6	0	12	4	3
I6	10	1	7	13	4
I7	10	0	7	12	2

Tabla 5.9 Resultados de las propiedades estructurales por iteración

Los defectos en la tabla 5.9 están relacionados con los módulos y la manera en que las interfaces de éstos trabajan con el resto del sistema.

Iteración	Parametrizada	Acoplamiento débil	Encapsulado
I1	3	0	7
I2	0	0	15
I3	0	0	15
I4	0	0	57
I5	0	4	72
I6	0	3	64
I7	0	3	57

Tabla 5.10 Resultados de las propiedades de modularidad por iteración

La tabla 5.10 identifica los defectos que implican una inadecuada documentación en el código, comentarios que aclaren o hagan indicaciones acerca de aspectos importantes de la implementación, o que por el contrario haya demasiados comentarios.

Iteración	Especificada	Documentada	Auto descriptiva
I1	0	2	3
I2	0	5	1
I3	0	6	0
I4	0	67	4
I5	1	72	13
I6	1	72	15
I7	1	24	3

Tabla 5.11 Resultados de las propiedades descriptivas por iteración

La gráfica de la figura 5.4 muestra la distribución de defectos mostrados en las tablas previas, donde se observa que en las categorías corrección y descriptiva son las de mayor incidencia en las últimas iteraciones. Lo cual implica deficiencias en relación a variables, su asignación, inicialización o utilización en estructuras de control y su descripción para hacer más clara la codificación.

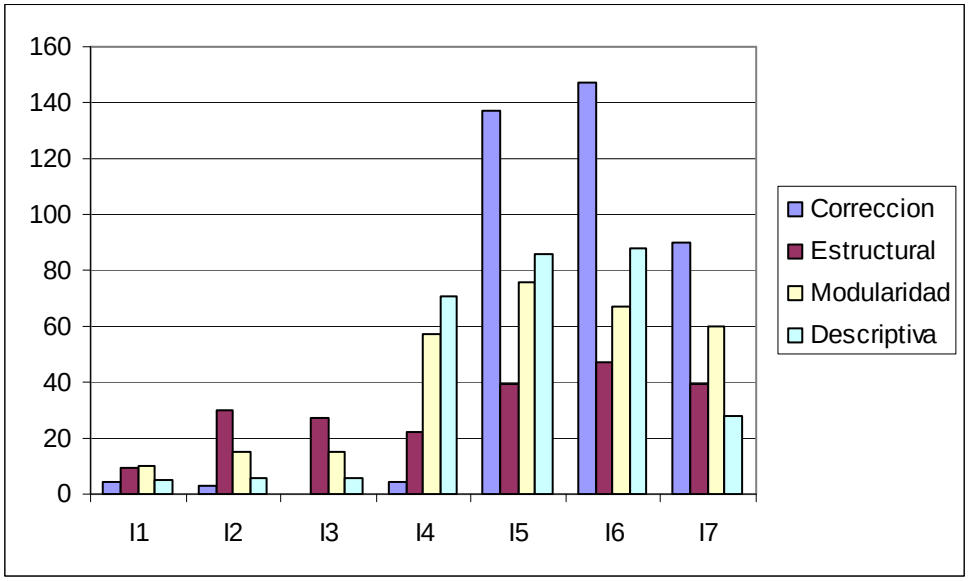

Figura 5.4 Gráfica de defectos por tipo

Aparte de revisar la calidad del proyecto, el propósito principal es identificar errores en las iteraciones, la figura 5.5 muestra la distribución de errores encontrados por iteración. En la figura 5.6 se desglosa el total de errores entre aquellos que se detectaron con el método tradicional (errores similares) y los que no fueron identificados anteriormente, sino por la aplicación del modelo generado (errores nuevos).

En promedio se localizaron alrededor de 24 errores nuevos, que no se lograron identificar con el otro método. Lo cual implica una ventaja, ya que independientemente que se hayan utilizado criterios distintos en cuanto a la búsqueda de errores, los errores nuevos implican problemas en la operación del software, que de algún modo quedaron ocultos con la revisión del método tradicional.

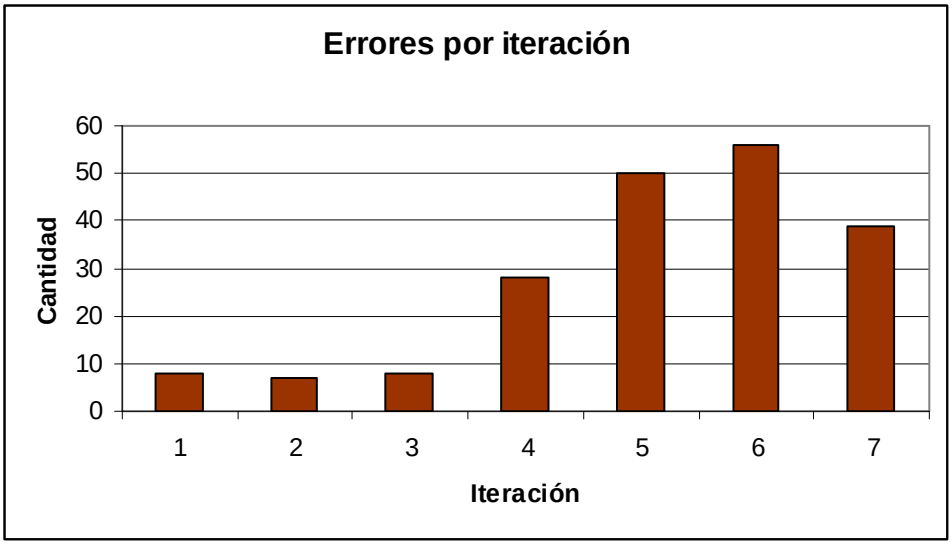

Figura 5.5 Gráfica de errores detectados con el modelo generado

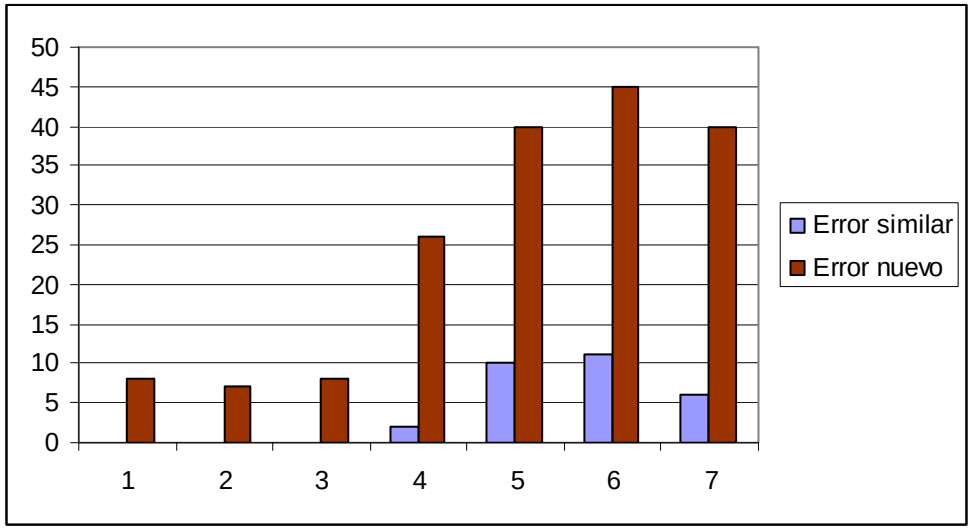

Figura 5.6 Errores similares y nuevos, detectados con el modelo generado

En la tabla 5.11 se muestra además del total de errores, el cálculo de la densidad de error, de donde se obtiene que el promedio de errores por iteración es de 29 y el promedio del porcentaje de densidad de errores es de 0.73%.

Iteración	Total de errores	%DE
1	8	1.45%
2	7	0.56%
3	8	0.60%
4	28	0.63%
5	50	0.77%
6	56	0.68%
7	46	0.42%

Tabla 5.12 Porcentaje de densidad de error con el modelo generado

Modelo	Promedio de errores	Promedio de % DE
Tradicional	16.5	0.67%
Generado	29	0.73%

Tabla 5.13 Comparación de resultado entre los dos modelos

Sin pretender hacer una corroboración de la hipótesis, en la tabla 5.13 se muestra un comparativo de los resultados obtenidos con ambos modelos, en base a la métrica seleccionada. Quedaría pendiente realizar cálculos estadísticos detallados que permitan realizar conclusiones sobre los mismos.

De los resultados se observa que el modelo generado en este trabajo incremente la densidad de error, se logra un incremento de .06 %. Esto implica que se tiene una mayor efectividad, no sólo para establecer un nivel de calidad en el proyecto, sino en la identificación de errores durante el desarrollo.

También se realizó un análisis para identificar cuantos defectos y errores que se encontraron en las iteraciones, permanecieron hasta la finalización del proyecto, que es una de las situaciones que se pretende disminuir al aplicar el Modelo W.

Se encontró que 208 defectos de los 242 totales en la última iteración, corresponde a las etapas previas y sólo el 14 % se generaron en la última iteración, como se muestra en la figura 5.7

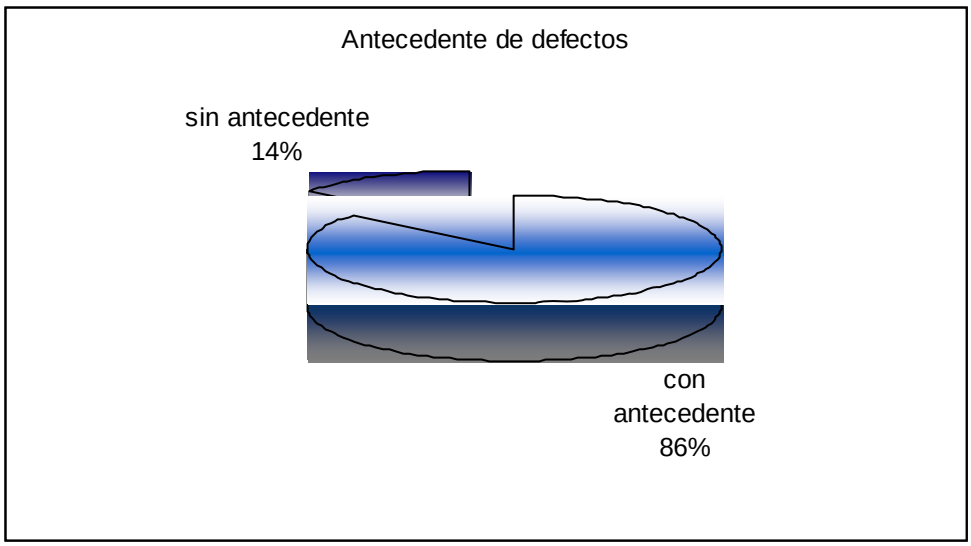

Figura 5.7 Porcentaje de defectos con antecedentes encontrados en la última iteración

Los errores que se acumularon durante las iteraciones se muestran en la figura 5.8, se puede observar que el 74% de estos tuvieron su origen a lo largo de las iteraciones. De haberse localizado y corregido previo a la finalización del proyecto, se tendría una densidad de error del 0.11% en vez de la que se obtuvo con el método tradicional que fue del 0.42%.

Lo cual comprueba que el modelo generado en este trabajo incrementa la localización de errores.

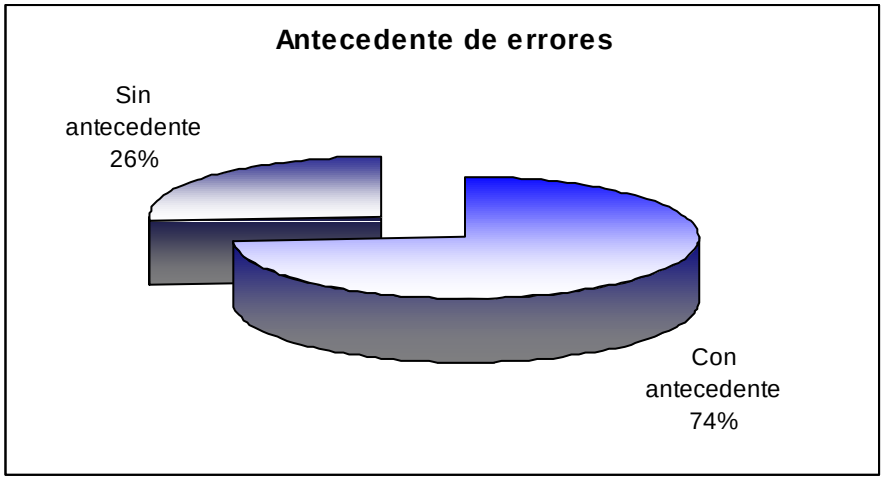

Figura 5.8 Errores generados en las iteraciones y que permanecieron

5.4 Impacto de defectos en la calidad del software

El marco de trabajo utilizado para evaluar la calidad del software, también permitió identificar aspectos que pueden conducir a una mala calidad en el producto final. A pesar de que estos defectos no implican necesariamente un mal funcionamiento de la aplicación, bajo el entorno y las condiciones en que se utiliza, puede llevar a inconsistencias si las condiciones cambian.

Para determinar qué aspectos de la aplicación pueden llegar a causar problemas, se hizo un análisis para determinar que atributos de calidad tienen más defectos. Se analizaron los defectos de la última iteración ya que no sufrió cambios posteriores, en la gráfica de la figura 5.9 se puede observar que de los 26 tipos de defectos que considera el modelo, se lograron detectar incidencias en 13.

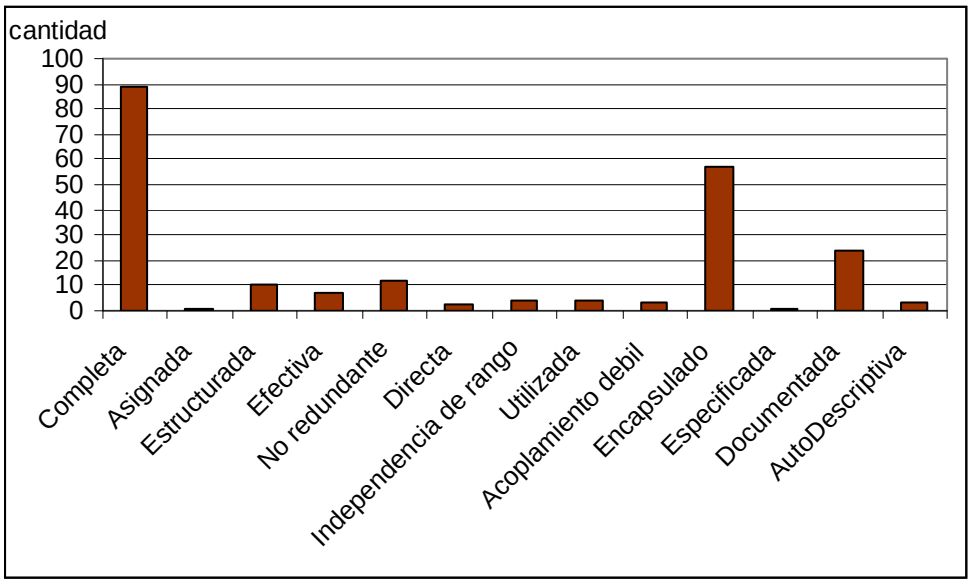

Figura 5.9 Incidencia de defectos en la última iteración

Tomando en cuenta la relación entre los tipos de defectos y los atributos de calidad que pueden ser afectados, se puede distinguir el atributo en el existe una mayor deficiencia y por tanto, determinaren que aspecto se podrían generar fallas. En la tabla 5.14 se muestra dicha relación y en la figura 5.10 se muestran los porcentajes de incidencia en los atributos de calidad.

	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenibilidad	Portabilidad	Reutilización
Completa	✓	✓	✓				
Asignada	✓	✓					
Estructurada	✓	✓			✓		
Efectiva			✓	✓	✓		
No redundante				✓	✓		
Directa				✓	✓		
Independencia de rango					✓		✓
Utilizada				✓	✓		
Acoplamiento débil		✓			✓	✓	✓
Encapsulado		✓			✓	✓	✓
Especificada	✓	✓	✓		✓	✓	✓
Documentada			✓		✓	✓	✓
Auto descriptiva			✓		✓	✓	✓

Tabla 5.14 Relación de defectos y atributos de calidad

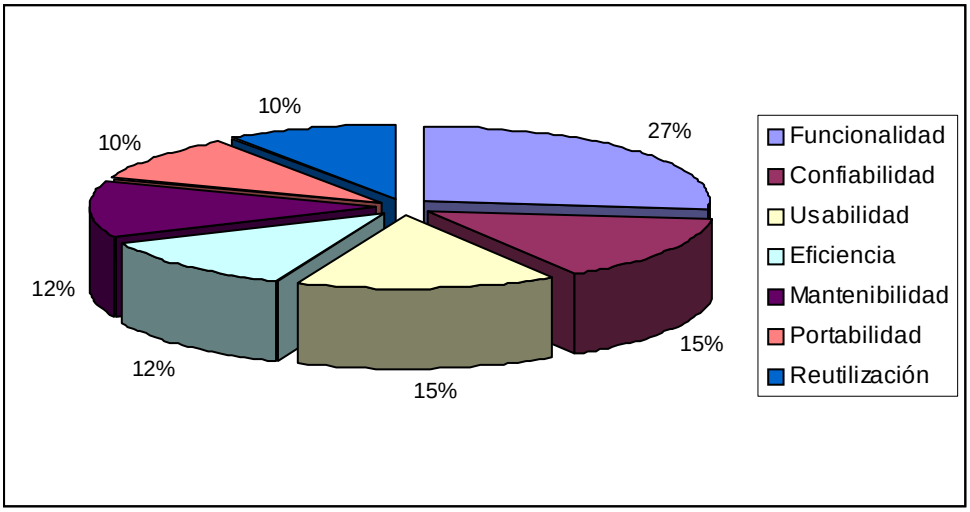

Figura 5.10 Gráfica de porcentaje de incidencias en atributos de calidad

De estos resultados se puede concluir que el impacto mayor es en los atributos de funcionalidad, confiabilidad y usabilidad, en menor medida en la eficiencia, mantenibilidad, portabilidad y reutilización.

5.4 Consideraciones finales

Realizado el análisis se encontró que la falta de procedimientos en la planeación, documentación y ejecución de pruebas, dificulta en gran medida la revisión del software en las iteraciones. Se puede caer en extremos de realizar procedimientos de prueba redundantes o de omisión y esto conducir a deficiencias en la detección y corrección de errores, impactando en la calidad del software.

Disponer de información cuantificable, permite evaluar con mayor precisión el producto de software y con ello poder aplicar estrategias que provean mayor calidad al término del proyecto. Como mostraron los resultados, una gran cantidad de defectos pudieron haberse detectado y corregido en etapas tempranas, lo cual no solo implica una mayor calidad sino también un menor esfuerzo en la aplicación de pruebas, mayor eficiencia en este proceso y finalmente un producto que además de cumplir con los requerimientos, pueda ser mantenible a un costo menor y con mayor rapidez.

Es importante recalcar que el principal objetivo en este capítulo era verificar la pertinencia de aplicar una mayor cantidad de pruebas en las iteraciones, como se propone en el Modelo W, más que evaluar el desarrollo en si, lo cual se comprobó.

Los resultados que se obtuvieron con el modelo propuesto, permitieron identificar una mayor cantidad de errores, además de probar que el modelo ayuda a la detección temprana de

errores. Se logró determinar qué aspectos de calidad, se ven afectados debido a la presencia de defectos y errores.

Cabe mencionar que las pruebas realizadas, únicamente consideraron las pruebas de unidad, lo que implica que al realizar pruebas en las otras etapas aumentaría la densidad de error y con ello maximizar aun más la eficiencia en la detección de errores.

Conclusiones

El principal objetivo de este trabajo era el de generar un modelo que pudiera aplicarse a un ciclo de vida de prototipo y que permitiera mejorar la detección de errores, se generó dicho modelo, al cual se le ha denominado Modelo W.

El modelo integra paradigmas de desarrollo de prototipo y de planeación, además de procesos de prueba que son incrementales y se aplican tanto a las iteraciones, como a las demás etapas del desarrollo. Con lo cual se cubren más elementos que pueden llegar a generar errores.

Se aplicó el Modelo W a un sistema real, al que se le habían aplicado previamente métodos de prueba, se compararon estos resultados con los obtenidos al aplicar el Modelo W y se obtuvo un .06% más en detección de errores, con lo cual se logró comprobar su efectividad, los resultados obtenidos permiten concluir lo siguiente:

- Al aplicar el modelo generado, se logra incrementar la detección de errores.
- Se detectan errores nuevos (que no se habían detectado con el método tradicional).
- Se pueden mejorar los atributos de calidad del producto de software.
- Se minimiza la propagación de errores y defectos durante el desarrollo de software.

La planeación, ejecución de pruebas y análisis de resultados que plantea el Modelo W, permite una mayor identificación de errores a lo largo del desarrollo, lo cual implica una mayor calidad en el producto final al minimizar errores y fallas en la implantación del software, que a su vez, aminora los costos de mantenimiento. Esto cumple el objetivo planteado, al hacer más eficiente la detección de errores en las iteraciones, se aminoran costos de mantenimiento e incrementa la calidad del software.

Además, se desarrolló un prototipo de software que apoya la sistematización del modelo, es una ayuda para la planeación de pruebas consideradas por el Modelo W, en todos los niveles de desarrollo y también permite una revisión constante del estatus de las pruebas.

Contribuciones

La principal contribución de este trabajo es haber adaptado las características de dos modelos de desarrollo, para la creación del Modelo W, el cual mejora la calidad de un producto de software al minimizar los errores al final de su desarrollo y por tanto mejorar la etapa de mantenimiento, conservando las características de mejor definición de requerimientos, aminorar el riesgo y tiempo en el desarrollo. Este modelo es una alternativa de desarrollo en aplicaciones que utilicen un modelo de prototipo incremental.

Otra contribución importante es haber mostrado la importancia de la planeación y ejecución de pruebas en un modelo de prototipo. Ya que se logró demostrar qué atributos de calidad se pudieron haber mejorado en la aplicación que se probó, si se hubieran aplicado más pruebas en las iteraciones, se pudieron haber eliminado errores y defectos de las etapas finales desde etapas tempranas, los costos de mantenimiento se podrían haber reducido.

Limitaciones

Una limitante es la del tiempo y recursos, para aplicar el Modelo W en un sistema real desde su inicio y así probar completamente todas las etapas que implica.

Tampoco fue posible probarlo en varios sistemas y así lograr un análisis concluyente respecto de la hipótesis planteada.

Trabajo futuro

Como trabajo futuro se propone desarrollar formatos de documentación para la información que puede generarse en el modelo, en este trabajo se proponen lineamientos generales de planeación de pruebas y resultados de las mismas, basados en estándares de la IEEE. Sin embargo, en la práctica se requieren lineamientos que precisen o detallen en mayor medida la información de pruebas según el tipo de desarrollo, para lo cual se requiere de una mayor investigación y considerar otros factores como la eficacia y el flujo de información entre los posibles involucrados en un proyecto y sus actividades.

También es importante investigar en lo relacionado a métricas que puedan ayudar a medir otros aspectos como el desempeño, la competencia, etc.

En cuanto al prototipo construido, solamente considera los aspectos de planeación del Modelo W, sin embargo es factible de mejorarse y automatizar otras actividades como la revisión del código fuente, el análisis de la información y generar indicadores en base a métricas.

Finalmente se deja como trabajo futuro continuar con la prueba de la hipótesis para llegar a una conclusión más amplia en su planteamiento.

Bibliografía

[1] M. G. e. a. Piattini Velthuis, *Análisis y diseño detallado de aplicaciones informáticas de gestión*. México: Alfaomega : Ra-Ma, 2000.

[2] G. J. Myers, "The art of software testing / Rev. Tom Badgett y Todd M. Thomas," vol. 2007, 2 rev. y aum. ed. Estados Unidos: John Wiley & Sons, Inc., 2004.

[3] M. Alavi, "An assessment of the prototyping approach to information systems development" *Commun. ACM* vol. 27 pp. 556-563 1984

[4] E. J. R. J. P. Dustin, *Automated software testing : introduction, management and performance*. Boston: Addison-Wesley, 1999.

[5] M. G. Piattini Velthuis, *Calidad en el desarrollo y mantenimiento del software*. México: Alfaomega, 2003.

[6] W. C. Hetzel, *The complete guide to software testing*. USA: John Wiley & Sons Inc., 1998.

[7] R. G. Dromey, "Concerning the Chimera [software quality]," *Software, IEEE*, vol. 13, pp. 33-43, 1996.

[8] N. Nagappan, L. Williams, M. Vouk, and J. Osborne, "Early estimation of software quality using in-process testing metrics: a controlled case study " en *Proceedings of the third workshop on Software quality* St. Louis, Missouri ACM Press, 2005 pp. 1-7

[9] "IEEE guide for the use of IEEE standard dictionary of measures to produce reliable software," in *IEEE Std. 982.2-1988*, 1989.

[10] "IEEE recommended practice for software requirements specifications," en *IEEE Std. 830-1998*, 1998.

[11] "IEEE standard dictionary of measures to produce reliable software," en *IEEE Std. 982.1-1988*, 1989.

[12] "IEEE standard for software test documentation," en *IEEE Std. 829-1998*, 1998.

[13] "IEEE standard for software unit testing," en *ANSI/IEEE Std. 1008-1987*, 1986.

[14] "IEEE Std. 982.1 - 2005 IEEE Standard Dictionary of Measures of the Software Aspects of Dependability," en *IEEE Std 982.1-2005 (Revision of IEEE Std 982.1-1988)*, 2006, pp. 0_1-34.

[15] "Industry implementation of International Standard ISO/IEC 12207: 1995. (ISO/IEC 12207 standard for information technology - software life cycle processes - implementation considerations," en *IEEE/EIA 12207.2-1997*, 1998.

[16] I. Somerville, *Ingeniería de software*, 6 ed. México: Pearson Educación, 2002.

[17] S. L. Pfleeger, *Ingeniería de software : teoría y práctica*. Argentina: Prentice Hall, 2002.

[18] E. J. Braude, *Ingeniería de software : una perspectiva orientada a objetos*. México: Alfaomega, 2003.

[19] "INTERNATIONAL STANDARD ISO/IEC 12207 SOFTWARE LIFE CYCLE PROCESSES."

[20] A. Toffler, *La tercera ola*. Barcelona: Plaza y Janés, 1999.

[21] S. H. Khan, *Metrics and Models in Software Quality Engineering*, 2a ed. ed. Boston: Pearson Education Inc., 2005.

[22] R. G. Dromey, "A model for software product quality," *Software Engineering, IEEE Transactions on*, vol. 21, pp. 146-162, 1995.

[23] I. Burnstein, "Practical Software Testing : A Process-oriented Approach," vol. 2007. New York: Springer Science & Business Media, 2003.

[24] R. Vonk, *Prototyping*. Gran Bretaña: Prentice Hall, 1990.

[25] O. Gutierrez, "Prototyping techniques for different problem contexts," en *Proceedings of the SIGCHI conference on Human factors in computing systems: Wings for the mind*: ACM Press, 1989, pp. 259-264.

[26] D. F. Rico, *ROI of software process improvement : metrics for project managers and software engineers*. Boca Raton, Florida: J. Ross, 2004.

[27] M. y. R. H. T. Dorfman, *Software Engineering*. USA: IEEE Computer Society Press, 1997.

[28] S. R. Schach, *software engineering*. Estados Unidos: Aksen associates incorporated, 1990.

[29] R. S. Pressman, *Software Engineering : a practitioner's approach*, 6 ed. Singapore: McGraw-Hill Education, 2005.

[30] R. N. Charette, *Software engineering environments : Concepts and technology*. New York, 1986.

[31] L. O. Ejiogu, *Software engineering with formal metrics*. Boston: QED Technical, 1991.

[32] D. Galin, *Software quality assurance : from theory to implementation*. Harlow, Essex ; New York: Pearson/Addison Wesley, 2004.

[33] S. Wagner and T. Seifert, "Software quality economics for defect-detection techniques using failure prediction " en *Proceedings of the third workshop on Software quality* St. Louis, Missouri ACM Press, 2005 pp. 1-6

[34] P. Goglia, *Testing client/server applications*. Boston: QED Publishing group, 1993.

Apéndice

A Descripción de Propiedades de Calidad

Descripción de propiedades de calidad agrupadas por categoría, se indica a que tipo de elemento de programación se aplica, el impacto en los atributos de calidad y también se dan algunos ejemplos de defectos. Basado en el "Modelo de calidad de producto de software" de R. Geoff Dromey[30].

Características de corrección

Computable	Aplica a: expresiones	Impacto en la calidad: funcionalidad, confiabilidad
Los resultados obedecen a leyes de la computación, aritméticas, etc.		
Defectos: ▪ División por cero. ▪ Script fuera de rango. ▪ Escribir en un archivo no abierto. ▪ División por una variable de estatus desconocido. ▪ Raíz cuadrada de un número negativo o de una variable de estatus desconocido.		

Completa	Aplica a: objetos, módulos, instrucciones	Impacto en la calidad: funcionalidad, confiabilidad, utilidad, mantenimiento
Si se tienen todos los elementos para implementarla y que satisfaga su propósito		
Defectos: ▪ Instrucciones if que pueden abortarse (específico del lenguaje). ▪ Auto asignaciones (x := x). ▪ Instrucciones de código que no se ejecutan en un mecanismo de selección. ▪ Módulos que no generan salidas.		

[30] Dromey, R. G. (1995). A model for software product quality. *Software Engineering, IEEE Transactions on*, 21(2), 146-162.

Asignada	Aplica a: variables	Impacto en la calidad: funcionalidad, confiabilidad
Si recibe un valor por asignación, entrada o parametrización		
Defectos: ▪ Uso de una variable en un término o expresión que no se le ha asignado un valor previamente.		

Precisa	Aplica a: variables y constantes	Impacto en la calidad: funcionalidad, confiabilidad
Precisión adecuada y preservada en los cálculos		
Defectos: ▪ Uso de precisión simple cuando los cálculos requieren doble precisión. ▪ Uso de un entero cuando el problema se encuentra en el rango de 0..9.		

Inicializado	Aplica a: Ciclos	Impacto en la calidad: funcionalidad, confiabilidad, mantenimiento.
Si todas las variables de un ciclo toman un valor antes de que comience su ejecución		
Defectos: ▪ Cuando las variables de un ciclo son sobre-inicializadas, prematuramente inicializadas.		

Progresivo	Aplica a: módulos (recursivos), ciclos	Impacto en la calidad: funcionalidad, confiabilidad, mantenimiento.
Un ciclo o estructura recursiva es progresiva si hay evidencia clara de que avanza hacia la terminación de sí mismo.		
Defectos: ▪ Ciclos anidados, donde las variables de salida del ciclo exterior solamente son modificadas en una ciclo interno (con una precondición) o por un llamado de función.		

Variante	Aplica a: condiciones (para ciclos y estructuras recursivas)	Impacto en la calidad: funcionalidad, confiabilidad, mantenimiento.
Si define una relación congruente y derivable de una función, que permite la terminación o cumplimiento de dicha condición.		
Defectos: ▪ Ciclos que utilizan una variable boolena como bandera en una condición (while not found do...) y no tienen una relación que derive de las funciones variantes dentro del ciclo. ▪ Ejemplo: un ciclo que tiene que tiene una función variable j-i-1 que decrece por medio de i:=i+1 y/o j:=j-1 una condición apropiada que es variante sería $i \neq j-1$)		

Consistente	Aplica a: módulos, instrucciones, condiciones, expresiones, variables y registros	Impacto en la calidad: funcionalidad, confiabilidad, mantenimiento, re uso, portabilidad, utilidad.
Cuando una estructura mantiene sus propiedades y/o funcionalidad y sus elementos contribuyen a reforzarla		
Defectos: ▪ Utilizar una variable para más de un propósito en un determinado ámbito. ▪ Modificar una variable de un ciclo al salir de éste. ▪ Utilizar una variable como constante. ▪ Cambiar una variable en una expresión. ▪ Entradas que no se utilizan (read(x);...; read(x)). ▪ Salida de una variable más de una vez sin cambio. ▪ Uso de variables/constantes de diferente precisión/tipo en cálculos.		

Propiedades estructurales

Estructurada	Aplica a: secuencias, condiciones,	Impacto en la calidad: mantenimiento, confiabilidad, funcionalidad.
Una sola entrada una salida		
Defectos: ▪ Salida a mitad de un ciclo. ▪ Múltiples regresos de una función. ▪ Condiciones de ciclo con demasiadas condiciones.		

Resuelta	Aplica a: ciclos	Impacto en la calidad: mantenimiento, eficiencia.
El control de la implementación de la estructura corresponde a la estructura de datos		
Defectos: ▪ Uso de un solo ciclo para procesar un arreglo bidimensional.		

Homogénea	Aplica a: ciclos, módulos (recursivo)	Impacto en la calidad: mantenimiento.
Solo invariantes conjuntivos para ciclos (A and B and...) A,B,... puede ser disyuntivo		
Defectos: ▪ Una estructura de ciclo cuya funcionalidad no es cohesiva (su predicado principal es disyuntivo).		

Efectiva	Aplica a: expresiones, instrucciones	Impacto en la calidad: utilidad, eficiencia, mantenimiento
Una forma estructural es efectiva si tiene todos, y únicamente aquellos, elementos necesarios para definirla e implementarla. Sin redundancia computacional		
Defectos: ▪ Asignaciones que establecen una misma condición previamente establecida. ▪ Expresiones con cálculos innecesarios y:=x+1+1.		

No redundante	Aplica a: condiciones	Impacto en la calidad: eficiencia, mantenimiento
Una forma estructural es no redundante cuando tiene todos, y solamente aquellos elementos, necesarios para definirla. Sin redundancia lógica		
Defectos: ▪ Probar una condición que ya se ha establecido.		

Directa	Aplica a: instrucciones, expresiones, variables, constantes, tipos	Impacto en la calidad: mantenimiento, eficiencia
Estructura directa, si el cálculo es directo, y éste será directo si la abstracción, forma de representación y estructura son congruentes con el problema original.		
Defectos: ▪ Uso de banderas (booleanas o de otro tipo). ▪ Uso de números para representar colores. ▪ Uso de trucos (Ej. Inicializar una matriz de identidad de la forma (I/J)*(J/I). ▪ Uso de variables booleanas para representar condiciones.		

Ajustable (parametrizable)	Aplica a: llamadas a módulos, expresiones	Impacto en la calidad: mantenimiento, re uso, portabilidad
Si no contiene constantes no declaradas, y el número de variables mínimas de propósito son usadas.		
Defecto: ▪ Si una forma estructural contiene números en vez de constantes definidas		

Independencia de rango	Aplica a: declaraciones (arreglos), ciclos	Impacto en la calidad: reuso, mantenimiento
Si sus límites inferior o superior no son constantes numéricas específicas. Un arreglo con límites fijos.		
Defecto: ▪ Un arreglo o variable se declara con límite fijo superior o inferior. ▪ Un ciclo que procesa un arreglo y asume que se inicia en 0 o 1.		

Utilizada	Aplica a: objetos, módulos, todas las formas de declaración de datos	Impacto en la calidad: mantenimiento, eficiencia
Una forma estructural es utilizada si es definida y usada dentro del mismo alcance.		
Defectos: ▪ Una variable o función que se declara y no se utiliza		

Propiedades de modularidad

Parametrizada	Aplica a: módulos	Impacto en la calidad: mantenimiento, re uso, portabilidad
Si contiene como parámetros solo los necesarios y suficientes de entrada y salida que la caracterizan como un procedimiento/función bien definido.		
Defectos: ▪ Módulos sin parámetros. ▪ Demasiados parámetros. ▪ Parametrización débil, aquella que modifica datos de entrada.		

Acoplamiento débil	Aplica a: llamadas a módulos	Impacto en la calidad: mantenimiento, re uso, portabilidad, confiabilidad
si sus llamadas son acopladas a través de datos (data-coupled)		
Defectos: ▪ Acoplamiento de control. ▪ Acoplamiento de estampado. ▪ Acoplamiento por contenido. ▪ Acoplamiento común. ▪ Externamente acoplado.		

Encapsulado	Aplica a: variables, constantes y tipos	Impacto en la calidad: re uso, portabilidad, confiabilidad
Una variable debe ser usada sólo dentro del alcance para el que fue definida.		
Defectos: ▪ Uso de variable en un módulo que no ha sido declarado dentro del ámbito del módulo		

Cohesivo	Aplica a: secuencias	Impacto en la calidad: mantenimiento, re uso, portabilidad
Si todos sus elementos están ligados entre sí y todos contribuyen a conseguir un solo objetivo. Las relaciones entre lo elementos de una entidad son maximizados		
Defectos: ▪ Un módulo con muchos parámetros tiene poca cohesión y probablemente implementa más de una función. ▪ Ciclos con inicialización dispersa.		

Genérico	Aplica a: módulos	Impacto en la calidad: mantenimiento, re uso, portabilidad.
Es dependiente del tipo de datos de sus entradas y salidas. Si sus cálculos son abstraídos a una forma de tipo parametrizado.		
Defectos: ▪ Dependiente de tipos primitivos (por ejemplo un procedimiento para intercambiar enteros swap(a,b)).		

Abstracto	Aplica a: Objetos	Impacto en la calidad: re uso, mantenimiento
Si no hay un concepto de alto nivel obvio que abarque la forma estructural		
Defectos: ▪ Módulos/objetos especializados (Ej. Declarar una clase de objeto carro, en vez de una clase de objeto vehiculo)		

Propiedades descriptivas

Especificada	Aplica a: objetos, módulos, ciclos, secuencias	Impacto en la calidad: funcionalidad, mantenimiento, confiabilidad, utilidad, portabilidad, re uso
Si todos los bloques, ciclos y funcionalidad incluyen sus variantes, precondiciones y postcondiciones.		
Defectos: ▪ La funcionalidad no esta descrita por precondiciones y postcondiciones. ▪ No dispone de precondiciones y postcondiciones. ▪ La especificación es ambigua, imprecisa, inconsistente o incompleta.		

Documentada	Aplica a: objetos, módulos, ciclos, secuencias, llamadas a módulos, estructuras de datos, variables, constantes y tipos	Impacto en la calidad: mantenimiento, portabilidad, re uso, utilidad
Si su propósito, estrategia, intención y propiedades son todas explícitas y bien definidas dentro del contexto de la estructura. Comentarios asociados a los bloques		
Defectos: ▪ Formas estructurales que no contienen comentarios. ▪ Insuficientes comentarios para describir propósito. ▪ Más comentarios de los necesarios. ▪ Documentación errónea.		

Auto descriptiva	Aplica a: objetos, módulos, llamadas a módulos, variables, constantes, estructuras de datos	Impacto en la calidad: mantenimiento, portabilidad, re uso, utilidad
Si su propósito, estrategia, intención, propiedades y tipo son evidentes en su nombre y los identificadores son congruentes con el contexto de la aplicación.		
Defectos: ▪ Nombre que no tienen relación con sus propiedades. ▪ Nombre innecesariamente largo. ▪ Nombre ambiguo o erróneo.		

Glosario

Atributo de calidad

Característica del software, que define un aspecto en particular de su funcionamiento, que puede ser evaluada y medida cuando se ejecuta el software.

Big bang

Estrategia de prueba que consiste en revisar exhaustivamente todos los módulos que componen un programa de cómputo.

Bottom up

Estrategia de prueba que realiza una revisión de ciertos módulos, iniciando con los de bajo nivel hasta llegar a los de más alto nivel.

Caja blanca

Prueba de software que verifica su funcionamiento interno o estructural.

Caja negra

Prueba de software que verifica aspectos funcionales del mismo.

Ciclo de vida del software

Etapas y secuencia que se consideran para elaborar un proyecto de software desde su concepción hasta su fin de uso.

Compatibilidad

Característica en determinado software para ejecutarse y funcionar en equipos y/o sistemas operativos distintos.

Complejidad ciclomática

Métrica que permite analizar la complejidad estructural del código a partir de los recorridos involucrados en una estructura cíclica.

Componente de software

Elemento de software con una funcionalidad particular para un entorno determinado, que puede formar parte de un desarrollo de software y que dispone de una implementación propia.

Defecto

Un proceso, definición de datos o una parte de procesamiento incorrectos en un programa.

Delphi

Lenguaje de programación de propósito general basado en una variación de pascal llamado Object Pascal, dispone de un entorno de desarrollo visual.

Densidad de error

Métrica que permite expresar la proporción de errores detectados respecto de las líneas de código.

Espacio de Aprendizaje (EA)

Aplicación para impartir cursos a distancia desarrollada en el Centro Universitario de Investigaciones Bibliotecológicas de la UNAM.

Error

Instrucción incorrecta de un programa.

Escalable

Característica del software que le permite adaptarse a situaciones cambiantes ya sea en el manejo de información, el equipo en el que se ejecuta o las modificaciones que se le hacen, sin que pierda calidad en su funcionamiento.

Estándar de software

Normas que permiten unificar, especificar y simplificar determinados procedimientos en diversos aspectos del software como la construcción, uso, implantación, etc., con la finalidad de obtener mejoras tanto en los procesos como en el entorno del software.

Estrés

Es una prueba de software que consiste en procesar un gran volumen de información, una gran cantidad de repeticiones o ejecución en condiciones de funcionamiento límite.

Falla

Incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados.

HTML (*HyperText Markup Language*)

Lenguaje de marcado basado en etiquetas que permite estructurar documentos con hipertexto y ser transferidos en Internet a través del protocolo HTTP y visualizarse por medio de navegadores.

IEEE (*Institute of Electrical and Electronic Engineers*)

Organización que desarrolla, define y revisa estándares en el área eléctrica, electrónica, de computación e informática.

Ingeniería de software

Conjunto de etapas parcialmente ordenadas con la intención de logra un objetivo, en este caso, la obtención de un producto de software de calidad.

ISO (*International Standard Organization*)

Es una organización no gubernamental que promueve el desarrollo de normas internacionales. Esta formado por una red de institutos de normas nacionales de diversos países.

Iteración

En el desarrollo de software basado en un modelo incremental o evolutivo, una iteración es cada incremento que se realiza en el desarrollo para aumentar la funcionalidad en el software hasta llegar al producto terminado.

LINUX

Sistema operativo desarrollado bajo el esquema de software libre, basado en UNIX para ser ejecutado en computadoras personales principalmente.

LOC (*Lines Of Code*)

Iniciales que indican el número de líneas de código de un programa fuente.

Metodología

Una serie de procesos a realizar, con un orden específico y que pueden incluir estándares.

Métricas de software

Mediciones de algún atributo del proceso de software.

Modelo de ciclo de vida de software

La serie de etapas relacionadas de manera particular y que se realizan en un orden específico, con las cuales se lleva a cabo la construcción de software.

Modelo W

Modelo de desarrollo de software basado en prototipo incremental modelo V que hace énfasis en la planeación y ejecución de pruebas de forma incremental.

Multiusuario

Sistemas que atienden los procesos de más de un usuario a la vez y comparten los mismos recursos.

MySQL

Sistema para el manejo de bases de datos relacionales multiusuario, se distribuye bajo el esquema de software libre y es desarrollado por la empresa MySQL AB.

ODBC. (*Open DataBase Connectivity*)

Interfaz de programación que permite a los clientes acceder a distintos tipos de sistemas de bases de datos y formatos de archivos.

OOP (*Object Oriented Programing*)

Programación Orientada a Objetos. es un método de implementación en el cuál los programas son organizados como grupos cooperativos de objetos, cada uno de los cuales representa una instancia de alguna clase, y estas clases, todas son miembros de una jerarquía de clases relacionadas bajo los principios de abstracción, encapsulación, modularidad y herencia, entre otros.

PHP (Acrónimo de *Hypertext Preprocessor*)

Lenguaje de programación basado en open source, es un lenguaje de tipo script que se ejecuta en el lado del servidor y puede ser embebido en páginas HTML, para crear aplicaciones dinámicas.

Portabilidad

Capacidad del software para poder ejecutarse en diferentes plataformas y/o arquitecturas con mínimas modificaciones o ninguna.

Prototipo de software

Es una representación limitada de un sistema de cómputo que refleja la funcionalidad esencial que debe realizar dicho sistema. Los prototipos permiten definir de forma más precisa los requerimientos de software.

Prototipo incremental

Desarrollo de software basado en la construcción de un prototipo y la generación de funcionalidad (llamada incremento), hasta llegar al producto de software final.

Sun Microsystems

Empresa informática que desarrolla servidores, estaciones de trabajo, el sistema operativo Solaris, el lenguaje de programación Java, entre otras cosas.

Top down

Estrategia de prueba que realiza una revisión de ciertos módulos, iniciando con los de más alto nivel hasta llegar a aquellos módulos de nivel más bajo.

Unitaria

Prueba que revisa principalmente la estructura del código fuente, ayuda a encontrar errores de lógica en el programa y las prácticas de programación.

UNIX

Sistema operativo multiusuario, multitarea, originalmente desarrollado en los laboratorios Bell, actualmente se tienen diversas versiones de este sistema.