Universidad de las Ciencias Informáticas

Facultad 10. Software Libre.

Título: “TÉCNICAS FORMALES PARA EL ANÁLISIS Y PREDICCIÓN DEL COMPORTAMIENTO FIABLE DEL SOFTWARE”

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Yaima Arocha Chang

Susel Cañete Pollán

Tutores: Msc. David Leyva Leyva

Ing. Daymy Tamayo Ávila

Ciudad de La Habana

Junio 2007

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al <nombre área> de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____	_____
Yaima Arocha Chang	Msc. David Leyva Leyva
_____	_____
Susel Cañete Pollán	Ing. Daymy Tamayo Ávila

AGRADECIMIENTOS

Agradecemos profundamente a todos los que de una forma u otra formaron parte de esta obra que se ha convertido en el fruto de cinco años de esfuerzo y dedicación.

De una forma muy especial a nuestro líder y Comandante en jefe Fidel que nos dio la posibilidad de forjarnos como ingenieros informáticos y escribir nuestra historia en medio de la batalla de ideas del pueblo cubano. Gracias Comandante por haber sido la luz que nos conectó al futuro, con tus ideas, tu valor y la confianza que depositaste en cada uno de nosotros.

Nos sentimos eternamente agradecidos con las personas a quienes acudimos para obtener información y que desinteresadamente nos brindaron un poco de su fe y su ayuda.

Agradecemos a Patricia y Ossany, por darnos una luz casi extinguida; a nuestros padres, por darnos una razón de seguir adelante cada día y ayudarnos en todas nuestras ocurrencias; a la profesora Graciela González Pérez y Fernando Peón que también pusieron su empeño en que las cosas salieran como deben ser.

Agradecemos también a los profesores de Ingeniería de Software y de Programación de la facultad 10, así como a los compañeros de clases y amigos que de alguna forma aportaron ideas en nuestra investigación.

Mil Gracias!!!

DEDICATORIA

Dedico la realización de este trabajo:

A mis padres, que con sus consejos me hicieron más fuerte en estos cinco años, convirtiéndome en lo que hasta hoy soy. A mi papá tan amable; a mi mamá tan enérgica. A ambos por quererme tanto.

A mi tía Marilín, que siempre me ayudó y a toda mi familia.

A mis amigos, los verdaderos, los que están y los que no.

Susel

Dedico la realización de mi trabajo de diploma:

A mis padres que han sabido llevarme siempre por el mejor camino, que han estado en mis momentos buenos y malos y hoy celebran conmigo el fruto de largos años de estudio y sacrificio. A mi madre por desvelarse por mis problemas y estar siempre a mi lado.

Le agradezco a mi papá cada regaño que me ha dado, e incluso cuando me habla fuerte porque en el fondo sé que lo hace todo por mi bien, le agradezco cada consejo, le doy gracias por haberme ayudado a decidir cuando no sabía aun si quería estudiar informática, y por cada minuto de su vida que ha luchado por estar junto a mí. A la familia hermosa que tengo, que me ha apoyado desde el principio hasta el fin, con sus ideas y con sus ansias de verme feliz. A mi abuela y siempre amiga que nunca me deja fallar y con sus lecciones me enseñó a mirar al futuro y me vio siempre en el lugar de una estudiante graduada de la universidad.

A mis amistades y a todas aquellas personas que de una forma u otra estuvieron conmigo cuando las necesitaba para darme más que apoyo, ánimo para seguir adelante y luchar por ver realizado el sueño que finalmente se cumple con la culminación de esta Tesis.

A todos ellos, que también forman las ideas que dejé plasmadas aquí, gracias por ser parte de mí.

Yaima

RESUMEN

En el transcurso de los últimos años la producción de software ha ocupado un lugar significativo en el ámbito económico mundial. Sin embargo estos productos presentan un problema común, no tienen suficiente calidad y uno de los factores que más influye es la fiabilidad. Con este trabajo “Técnicas Formales para el análisis y la predicción del comportamiento fiable del software” se pretende contribuir a aumentar la fiabilidad en la producción del software en la Universidad de las Ciencias Informáticas.

Para ello, se estudiaron las técnicas más utilizadas actualmente en el desarrollo del software fundamentalmente las técnicas formales, así como las herramientas orientadas al trabajo con ellas. Además se analizaron algunas métricas que se utilizan para prevenir los fallos en el sistema.

Finalmente se presentó una propuesta que consiste en la aplicación de técnicas formales, herramientas y métricas que proporcionarán que el software en la UCI aumente su fiabilidad.

PALABRAS CLAVE

Fiabilidad, técnicas formales, herramientas, métricas, desarrollo de software.

ÍNDICE

INTRODUCCIÓN

Por nuestros días el incremento de las tecnologías de la información, es de gran relevancia a la hora de analizar y demostrar la fiabilidad de un software, ya que el objetivo principal de éste es crear un sistema que satisfaga las necesidades de los clientes y de los usuarios, por lo tanto es de suma importancia garantizar la calidad haciéndose necesario disponer de un proceso de desarrollo que garantice la seguridad máxima en dicho software, y a pesar de utilizar nuevas técnicas informáticas, el producto no siempre tiene calidad total.

Entre el software desarrollado y su entorno existe una estrecha relación, ya que éste es introducido en el mundo para provocar ciertos efectos en él y es ahí donde los clientes observarán si el desarrollo del software ha cumplido su propósito. En la mayoría de los casos estos productos carecen de calidad tanto en el propio proceso de desarrollo como en los productos que se entregan. Este problema tiene su origen en las habituales desviaciones de plazos y esfuerzo sobre los valores previstos, lo que pone de manifiesto la falta de calidad en el proceso de gestión de los proyectos de software: cuanto menor es ésta, peor es el grado de adherencia a los plazos y esfuerzos previstos.

La seguridad es un aspecto que se ha convertido en preocupación para quienes trabajan directamente con el software, esto pasa porque desde el momento que se viola su integridad, no es considerado un buen producto. También está en juego el prestigio de sus creadores. Pero un software no es totalmente seguro ya que cada día aparecen nuevos medios para penetrar en los sistemas, convirtiéndose éstos en más vulnerables.

Los riesgos de seguridad siempre están presentes, aunque en su mayoría ocurren por descuidos humanos que pueden ser: acceso al sistema por un terminal encendido, obtener la contraseña de otros usuarios, búsqueda en la basura para obtener información a partir de ficheros eliminados, etc. [Valido, 2006]

Para contrarrestar esas dificultades es que se crean nuevas expectativas, lo que conlleva al desarrollo de programas cada vez más complejos nada fáciles de construir.

El software es un entorno sistémico, donde los componentes deben encajar y funcionar armónicamente, alineados con las características, cultura y estrategia de la organización, para maximizar la homogeneidad y calidad de los resultados. El éxito o fracaso de las organizaciones que esperan un alto grado de calidad

y que trabajan sin metodologías depende del conocimiento tácito de su personal, pero hay que tener en cuenta que este conocimiento puede ser mucho o poco, bueno o malo, es decir, trabajar sin utilizar determinada metodología implica acudir a manuales y libros que en un final puedan dar la idea de lo que se quiere saber y no todo el mundo lo entendería de la misma manera.

Muchos ingenieros utilizan las técnicas tradicionales para asegurarse de que su producto será de máxima calidad, en cambio hay otras personas que prefieren utilizar las llamadas técnicas formales, las cuales, permiten verificar y validar el sistema en todas las fases del proceso de desarrollo, reduciendo el riesgo de propagar errores a lo largo de dicho proceso. Sin embargo, y a pesar de las ventajas potenciales de los métodos formales, es escasa su integración en la industria del software y esto se debe al desconocimiento de estas técnicas; la inexistencia de herramientas apropiadas que permitan su aplicación práctica; así como su poca utilización en el proceso de desarrollo software.

Durante los más de cincuenta años de historia de la informática ésta ha experimentado un desarrollo indudable, sin embargo la fiabilidad sobre bases ingenieriles sólidas, sigue constituyendo un reto que requerirá el esfuerzo combinado de muchos grupos de investigación y equipos de producción, es decir, la escasa fiabilidad en la construcción del software, se ha convertido en una necesidad reconocida a nivel mundial. Los requerimientos de los programas actuales precisan de herramientas potentes para obtener un producto seguro. Son necesarias, por tanto, técnicas y métodos que garanticen la calidad del proceso de desarrollo del software.

En la Universidad de las Ciencias Informáticas[1] se trabaja en grupos de proyectos que no quedan ajenos a la situación antes planteada. Como es objetivo del recinto universitario que la producción sea de óptima calidad, se hace imprescindible una profunda investigación sobre las técnicas más actuales y eficientes para la construcción del software en los proyectos de dicha facultad.

Por tanto el **problema científico** de este trabajo está basado en ¿Cómo contribuir a mejorar la fiabilidad del software que se desarrolla en la universidad? El **objeto de estudio**, se enmarca en la fiabilidad del software, teniendo como **campo de acción** las técnicas formales para la fiabilidad del software.

Las **preguntas científicas** propuestas son:

[1] Universidad de Ciencias Informáticas: UCI

- ¿Qué métodos se emplean para medir la fiabilidad del software en la Universidad de las Ciencias Informáticas?
- ¿Qué técnicas se utilizan actualmente para asegurar la calidad de un producto en los proyectos de la universidad?
- ¿Qué técnicas formales pueden ser aplicadas en los proyectos productivos de la universidad?

El **objetivo general** de este trabajo es hacer una propuesta de técnicas formales de verificación y validación para garantizar la fiabilidad del software, que sea aplicable a los proyectos informáticos que se desarrollen en la UCI. Para esto se hace necesario abordar las siguientes **tareas de investigación**:

- Profundizar en temas de calidad del software.
- Estudiar las técnicas formales aplicadas en el proceso de desarrollo del software que garanticen la fiabilidad del mismo.
- Profundizar en la búsqueda de herramientas actuales y más usadas orientadas a las técnicas formales para garantizar la calidad del producto.
- Analizar las métricas que posibilitan medir aspectos de fiabilidad del software.
- Realizar una propuesta de técnicas, herramientas y métricas que contribuya a garantizar la fiabilidad del software producido en los proyectos productivos en la UCI.

Para realizar la investigación se usaron métodos como el analítico-sintético que sirvió de marco para un profundo análisis de los temas que motivaron el trabajo. Para ello se realizaron búsquedas minuciosas tanto de calidad, haciendo énfasis en la fiabilidad, así como de las técnicas formales y herramientas para garantizar la misma, extrayendo sus aspectos más importantes.

También se hizo uso del método inductivo-deductivo porque son analizados les temas de investigación de forma general para arribar puntos específicos dentro de la investigación. Por ejemplo, al tratar los temas de la calidad y sus factores, haciendo énfasis particularmente en la fiabilidad.

Desde el momento del planteamiento del objeto de estudio y su respectivo campo de acción se hace uso del método genético, visualizando el campo como una célula puntual del objeto.

En cuanto a los métodos empíricos, se hizo uso de las entrevistas, que fueron útiles por el enriquecimiento del conocimiento adquirido con personalidades experimentadas en el ámbito del software. Además se aplicó una encuesta con el propósito de determinar gradualmente el comportamiento de la calidad del software en la universidad.

El documento está estructurado en tres capítulos y las secciones del glosario de términos, bibliografía y anexos.

En el Capítulo I se presentan los términos de calidad y los factores correspondientes. Además se da una primicia de las técnicas tradicionales y formales que servirán para la comprensión de la lectura del documento.[2]

El Capítulo II está centrado en la fiabilidad vinculando su integración en el proceso de desarrollo de software. Además se realiza una sistematización teórica de las técnicas formales utilizadas en cada etapa del ciclo de vida del software, así como las herramientas que contribuyen al desarrollo fiable del software. También se analizaron métricas a través de las cuales se puede medir la fiabilidad del software.

Para el Capítulo III se analizan los resultados de la investigación teniendo como referencia las encuestas y entrevistas realizadas. Se propone el uso de las técnicas expuestas en el capítulo 2 para ser aplicadas en los grupos productivos de la UCI y finalmente se proponen métricas que permiten evaluar la fiabilidad del software producido en la universidad.

[2] Los conceptos aparecerán textualmente porque es considerado necesario.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 INTRODUCCIÓN

Durante el desarrollo del presente capítulo se exponen los principales conceptos relacionados con la calidad del software, además serán vistos otros que de una forma u otra están estrechamente vinculados. También se hace mención de los factores que intervienen en la calidad en el proceso de desarrollo del software, siendo éstos un punto de partida que determinan el campo en el cual se desarrolla este trabajo: la fiabilidad.

Se tratarán de forma general algunos conceptos referidos a las principales técnicas tradicionales y formales que se utilizan actualmente en la construcción de software.

1.2 BREVE RESEÑA HISTÓRICA

Las técnicas o métodos formales, llegan al mundo después de la aparición de la Ingeniería de Software (IS) como parte de ella. En 1968 durante la conferencia de la OTAN[3] en Garmish (Alemania), se trataron varios temas, específicamente sobre la construcción del software. [Barceló, 2006]

En este congreso se habló de lo difícil que era y el trabajo que se pasaba cuando se iba a construir un software completamente nuevo, seguro, fiable y de calidad. Pasaron varios años cuando por los 70, aparecen los primeros indicios de estos métodos formales, cuando aún no existían estándares ni modelos de calidad como CMM. Con la aparición de técnica Z, como una de las primeras, que dio origen a lo que sería después B. Estas surgieron precisamente para tratar de resolver los problemas que se venían sucediendo con el software, muchos errores que sin poderlos detectar a tiempo se convertían en serios problemas tanto para los ingenieros como para los usuarios finales. Debía buscarse una solución que acabara definitivamente con estos males.

Entonces se plantearon la meta de desarrollar sistemas de forma tal que podrían verificarlos, corregirlos, cuantas veces fuera necesario y en cualquier momento. Para ello establecieron métodos que

[3] Organización del Tratado del Atlántico Norte

matemáticamente o mediante reglas lógicas, definían un sistema y podían comprobarlo disminuyendo así los índices de errores que tenía.

Poco a poco fueron desarrollando nuevos métodos que se dedicaban a ciertas ramas específicas de la industria, ganándose así la confianza de grandes productores. A pesar de las ventajas que representaban para la confiabilidad del software muchos ingenieros no se atrevían a asumirlos por sus bases matemáticas, alegando que para trabajar con ellos hay que tener conocimientos muy profundos de esa ciencia. Otros que antes se veían intimidados, tuvieron que aventurarse por problemas de seguridad o pérdidas muy grandes en sus empresas.

Actualmente lo que se conoce de los métodos formales es que sirven de gran ayuda para los sistemas de comunicaciones, compiladores, aplicaciones de transporte, etc. Aún así no es suficiente para que se utilicen en todos los países, ni en todas las industrias. No se deja de reconocer que es necesario saber de implementos matemáticos para trabajar con ellos, pero debiera hacerse un espacio a la oportunidad para aplicarlos porque es más provechoso su uso que tener pérdidas por no utilizarlos.

Esta forma de trabajo ha trazado un camino, el cual ha constituido un paso significativo para la calidad de sistemas informáticos, de modo que ésta se convierta en estudio obligatorio para la investigación.

1.3 CALIDAD

La calidad se ha convertido en un anhelo de cada ingeniero cuando realiza un software. Ésta ha pasado por varias fases y cada día se van perfeccionando los mecanismos que la identifican.

Puede verse en la tabla 1 (ver Anexo # 1) evidencia de la evolución histórica de la calidad y esencia de sus actividades propuesta en [Miranda, 2006]

Para comprender esos datos será necesario saber los conceptos básicos que a continuación se presentan, además que sirven de fundamento para la comprensión del documento en general.

1.3.1 Conceptos Básicos

Antes de comenzar a hablar de software es evidente preguntarse ¿Qué es la calidad?

“El concepto de calidad proviene del latín *Qualitas* y está asociado al atributo o propiedad que distingue a las personas, bienes o servicios”. [Miranda, 2006]

Según el Dr. Edward Deming, “la **calidad** es un producto o servicio consistente y confiable que satisface o excede los requerimientos del cliente al precio que está dispuesto a pagar”. [Instituto, 1999]

Es **calidad** también “el grado de cumplimiento de los requisitos y expectativas planteadas y aumento de la productividad.” [García, 2000]

En [Miranda, 2006] varios criterios señalan que:

“**La calidad** es el nivel de excelencia que la empresa ha escogido alcanzar para satisfacer a su clientela clave”, es decir, que debe estar encaminada a la satisfacción plena y total del cliente. También es el “grado en el que el conjunto de características inherentes cumple con los requisitos”

De estos conceptos se consideran más acertados los propuestos en [Miranda, 2006] por ser más actuales y centrarse específicamente en la satisfacción del cliente como el objetivo primordial, y para esto proponerse cumplir incondicionalmente con sus especificidades. Sin dejar de reconocer los aportes de Deming quien fue uno de los más grandes estudiosos del tema demostrando que cuando no se tienen establecidos planes concretos de calidad, los costos son más elevados por desperdicio de recursos, tiempo y tratar de eliminar reiteradamente los errores. [González, 2007]

La **gestión de la calidad** es el conjunto de acciones planificadas y sistemáticas, necesarias para dar la confianza adecuada de que un producto o servicio va a satisfacer los requisitos de calidad. [Portal, 2006]

En términos empresariales, no resulta tan fácil como parece regirse por un plan de gestión de calidad, sin embargo se hace imperante por la competencia del mercado que cada día se acrecientan más las metas a cumplir por exigencias del cliente.

1.3.2 Calidad Total

La calidad total es una estrategia de gestión que se pone de manifiesto en organizaciones para cumplir con expectativas de todo el personal involucrado. Está basada fundamentalmente en satisfacción de la alta dirección, del personal que desarrolla el sistema y del usuario final. Esto quiere decir que pretende unificar a todos con el propósito de trabajar con calidad. [Portal, 2006]

La calidad total es un proceso en evolución continua y que por su naturaleza misma no se puede detener, de lo contrario deja de ser un proceso. [Instituto, 1999]

1.3.3 Normas Internacionales

Cualquiera puede preguntarse cómo llevar a cabo un plan de esta envergadura. Son precisamente las normas internacionales las que contribuyen a que esto sea posible. Son por lo general, las que dirigen la línea organizativa de las compañías y la estandarización de calidad de los productos.

ISO es la Organización Internacional de Estandarización (en algunos casos Organización Internacional de Normalización) y fue creada para medir y asegurar la calidad en la producción. [Normas, 2007] Suele llamársele así por ser las siglas en inglés de International Standards Organization.

Teniendo en cuenta su relevancia para la realización de este trabajo, serán expuestos algunos ejemplos de las normas más utilizadas en el mundo informático.

La serie ISO, una de las más importantes, provee normas para servicios y productos, en todos los campos del hacer humano (Ingeniería, Biología, Medicina, etc). [Instituto, 1999]

ISO 9000 Fue desarrollada por la ISO con sede en Ginebra. y constituye un esquema integrador de esfuerzo de calidad, el cual permite la amortización a escala internacional de la calidad como elemento imprescindible en los intercambios comerciales. Estas normas son un conjunto de buenas prácticas de calidad en la realización y obtención de un producto o un servicio, entre ellos: [Instituto, 1999]

- Garantizan que un proveedor tiene la capacidad de producir los bienes y/o servicios requeridos, satisfaciendo las expectativas de los clientes.
- Facilita y promueve la actividad comercial e industrial.
- Impulsa a los trabajadores de la organización a un mejoramiento sucesivo.
- Optimiza las operaciones y procesos elaborados en la organización (se incrementa la eficiencia).
- Simplifica significativamente los costos ya que elimina desperdicios e ineficiencias de los sistemas y procesos.

- Fortalece la imagen de la empresa.

ISO 9001, es el modelo para el Aseguramiento de la Calidad aplicable a la producción, instalación y servicio posventa. [Instituto, 1999]. Según [Portal, 2006] la ISO 9001:2000 es la versión más actual y ha sido adoptada como modelo a seguir para obtener la certificación de calidad.

La especificación ISO 9001:2000, en su modelo propuesto de la norma ISO 9001 para el año 2000, es sin lugar a dudas, una evolución natural de las demandas de las organizaciones públicas y privadas para contar con herramientas de gestión más sólidas y efectivas para hacerse al incierto mar de la globalización y capitalizar sus esfuerzos. [Portal1, 2006] Como es una versión de la 9001, se rige por sus principios básicos, que serán vistos a continuación.

1.3.4 Principios Básicos de Calidad

Se siguen 8 principios básicos fundamentales de la gestión de calidad o excelencia. Se utilizan por la directiva de la entidad como un marco de referencia para guiar a las organizaciones hacia la consecución de la mejora del desempeño. Pueden encontrarse en [Portal, 2006] estos principios:

1. Organización enfocada a los clientes: las organizaciones dependen de sus clientes y por lo tanto comprender sus necesidades presentes y futuras, cumplir con sus requisitos y esforzarse en exceder sus expectativas.
2. Liderazgo: Los líderes establecen la unidad de propósito y dirección de la organización. Ellos deben crear y mantener un ambiente interno, en el cual el personal pueda llegar a involucrarse totalmente para lograr los objetivos de la organización.
3. Compromiso de todo el personal: El personal, con independencia del nivel de al organización en el que se encuentre, es la esencia de la organización y su total implicación posibilita que sus capacidades sean usadas para el beneficio de la organización.
4. Enfoque a procesos: Los resultados deseados se alcanzan más eficientemente cuando los recursos y las actividades relacionadas se gestionan como un proceso.
5. Enfoque del sistema hacia la gestión: Identificar, entender y gestionar un sistema de procesos interrelacionados para un objeto dado, mejora la eficiencia y la eficacia de una organización.

6. La mejora continua: La mejora continua debería ser el objetivo permanente de la organización.
7. Enfoque objetivo hacia la toma de decisiones: Las decisiones efectivas se basan en el análisis de datos y en la información.
8. Relaciones mutuamente beneficiosas con los proveedores: Una organización y sus proveedores son independientes y una relación mutuamente benéfica intensifica la capacidad de ambos para crear valor y riqueza.

1.3.5 Calidad del Software

Una vez tratados los criterios de calidad puede abrirse paso a conocer qué es la calidad del software.

Calidad es la capacidad del producto software para permitir que usuarios específicos logren realizar tareas específicas con productividad, efectividad, seguridad y satisfacción, en determinados escenarios de uso. El objetivo de un producto es que posea la calidad necesaria y suficiente para que satisfaga las necesidades del cliente explícitas e implícitas. [González, 2006]

La calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad. [Fernández, 1995]

El software tiene calidad cuando, además de ser útil para los usuarios, ha de ser eficiente en el uso y en el aprovechamiento del hardware y no ha de incorporar errores; por último, también conviene que tenga un mantenimiento fácil. [Barceló, 2006]

De estos enunciados puede inferirse que para que un producto tenga calidad, debe responder a las necesidades del cliente, no presentar errores, que sea seguro y de fácil entendimiento con el usuario.

1.3.6 Tipos de Calidad del Software

En términos de software existen dos tipos de calidad en los sistemas: [Calidad, 1999]

Calidad de diseño: Son especificaciones, características, programas específicos, procedimientos, etc., que prometen producir un servicio vendible que el cliente requiere, calidad de servicio esperada, implica el servicio más útil.

Calidad de conformación: Es la medida de la eficiencia en el logro de los resultados por la calidad esperada, prometida. Calidad de servicio resultante implica mayor confiabilidad de trabajo.

1.3.7 Control de la Calidad del Software

La IS juega un papel gestor en la calidad de los productos informáticos. Existen varias formas de conseguir calidad; estas son dos de las más conocidas según [Pavón, 2004]:

1. Haciendo Software Requirements Specification (en español, especificación de requisitos de software), diseños e implementaciones correctos desde un punto de vista técnico.
2. Introduciendo en el modelo del proceso una serie de actividades que garanticen que todas las entregas resultantes de una actividad de trabajo sean correctas.

Las técnicas de IS para conseguir calidad en el software se denominan Garantía de Calidad del Software (GCS), o en inglés, Software Quality Assurance (SQA). [Pavón, 2004]

La GCS engloba:

- Enfoque de gestión de calidad.
- Tecnologías de IS (métodos y herramientas).
- Revisiones Técnicas Formales durante el proceso de software.
- Estrategia de pruebas.
- Control de la documentación del software y de cambios.
- Procedimientos que aseguren un ajuste a los estándares de IS.
- Mecanismos de medición y generación de informes.

Controlar la calidad son una serie de revisiones, inspecciones y pruebas utilizadas a lo largo del proceso del software para asegurar que cada producto cumple con los requisitos que le han sido asignados. [Pavón, 2004]

1.3.8 Normas de la Calidad del Software

Las normas ISO 9000-3

Inicialmente la fabricación de software estaba incluida dentro de las normas 9001, normas que definían un modelo de garantía de calidad en diseño/desarrollo, producción, instalación y servicio.

Las características especiales y peculiares de la fabricación de software en que prácticamente todo el esfuerzo se dedica al diseño/desarrollo, y en que la influencia de la fabricación es prácticamente irrelevante, el comité ISO decidió establecer un modelo específico para la aplicación de las normas ISO 9001 para el desarrollo, suministro y mantenimiento de software: la norma 9000-3.

En La Universidad de Lleida, a finales de 1997 y principios de 1998, en la realización de un proyecto en equipo, se puso en práctica ISO 9000-3 vinculada a un caso práctico para el desarrollo de un software. Éstos, siguiendo una metodología explicada en detalles en [Lorés, 2006], obtienen resultados favorables en el tiempo planificado.

CMM y CMMI

Por otra parte, mundialmente se conocen las normas o modelos de calidad conocidos como SW CMM (Software Capability Maturity Model o en español, Modelo de Madurez de La Capacidad de Desarrollo del Software) que es el modelo más conocido en la industria del software en los países que lo desarrollan.

El modelo mide la capacidad del proceso para desarrollar un software con calidad, incrementando la predictibilidad para terminar los proyectos en coste, tiempo y con la calidad que el cliente espera. El modelo fue creado en 1986 por el Instituto de Ingeniería de Software (SEI según sus siglas en inglés) a solicitud del Departamento de Defensa de los Estados Unidos. Después de varias modificaciones que se le hicieron, fue finalmente en 1993, cuando se obtuvo la última versión del CMM (versión 1.1) según [Ema, 2002]. Su enfoque, está orientado a generar e implantar las mejores prácticas de ingeniería de software como producto principal.

CMMI (CMM Integrated, o Modelo de Madurez de Capacidad Integrado) es otro modelo derivado del CMM. CMMI tiene el propósito de proporcionar una única guía unificada para la mejora de múltiples

disciplinas tales como ingeniería de sistemas, ingeniería del software y el desarrollo integrado del producto y del proceso.

Las mejoras del CMMI sobre el SW-CMM

Hay que considerar que la base del CMMI está basada en el SW-CMM, por tanto tiende a superar algunas de sus limitantes. Sin embargo en [Almeraz, 2002] se plantea que no todas las empresas aceptan al CMMI como la mejor opción porque no se adapta a sus necesidades. Tal es el caso de las medianas y pequeñas empresas que se basan solamente en el desarrollo de software y no a la integración hardware, que es el enfoque principal del modelo CMMI. Esto quiere decir que el CMMI está destinado fundamentalmente a grandes compañías.

Debido a estas particularidades del CMMI respecto al SW-CMM, en [Almeraz, 2002] se plantea una real dificultad la aplicación de este modelo en empresas donde no se tiene experiencia con el SW-CMM o donde sólo se han aplicado los niveles inferiores del mismo.

Aunque el CMMI sigue algunas de las actividades del SW-CMM, se proyecta nuevos cambios en cuanto al enfoque y al alcance de sus objetivos. En [Ema, 2002] pueden encontrarse algunas de estas mejoras del CMMI que lo convierten en un modelo sólido para ayudar a obtener la calidad del software. Algunas son:

Se han agregado las áreas de proceso para hacer un mayor énfasis sobre algunas prácticas importantes:

- Medición y análisis.
- Gestión de riesgos.
- Análisis sistemático y la puesta en práctica de las decisiones acordadas.
- Todas las nuevas áreas de proceso definidas en la categoría de Ingeniería.
- Nuevo énfasis sobre el producto, así como sobre los procesos, incluyendo las interacciones con el cliente.
- Mayor importancia, desde las fases primeras, del análisis y la medición de los procesos empresariales.
- Cobertura de servicios, así como de sistemas.

- Especial énfasis sobre la capacidad de los procesos y madurez de la organización en su conjunto (no exclusivamente en el área de ingeniería del software).
- Mejor cobertura de la gestión de ingeniería integrada.
- Existe un nuevo enfoque de la formación. La educación y el entrenamiento adecuado para la mejora de la eficacia y de la eficiencia.
- Favorece el establecimiento de un ambiente adecuado para la gestión de los cambios dentro de la organización.
- Proporciona compatibilidad con los principios, requisitos y recomendaciones de la nueva norma ISO 9000:2000.
- Sienta las bases para que las organizaciones del sector de desarrollo del software se encaminen hacia el ciclo de la mejora continua.
- Entre otras.

1.4 FACTORES DE LA CALIDAD DEL SOFTWARE

Cuando se habla de calidad del software se debe diferenciar la calidad del producto y la del proceso de desarrollo de éste. Los objetivos de calidad que se establecen en el proceso de desarrollo del software se refieren a las metas que se establecen para determinar la calidad del producto.

La calidad está determinada por un grupo de factores que se subdividen en tres grandes grupos: [Cueva, 1999]

1. **Operaciones del producto**: Representa la característica operativa que tiene el producto y dentro de ella se encuentra:
    - Corrección
    - Fiabilidad
    - Eficiencia
    - Integridad

- Facilidad de uso

2. **Revisión del producto**: Determina la capacidad para soportar cambios que se hagan en el software.
    - Facilidad de mantenimiento
    - Flexibilidad
    - Facilidad de prueba
3. **Transición del producto**: Determina la adaptabilidad del software a nuevos entornos.
    - Portabilidad
    - Reusabilidad
    - Interoperabilidad

Los factores de calidad, además, se subdividen en criterios, de esta forma la calidad del software se puede medir a través de atributos que representan diferentes visiones de la misma.

1.4.1 Factores Internos

Los factores internos como modularidad y legibilidad, son aquellos que sólo son percibidos por aquellas personas que tienen acceso al código fuente; además están los factores externos que se analizarán a continuación y que en un principio son los que importan aunque la clave para conseguirlos radica en factores internos.

1.4.2 Factores externos

La **corrección**, como factor, es la capacidad que tiene el producto para realizar con exactitud todas sus tareas de forma correcta tal y como se define en las especificaciones.

Se define como la cualidad principal de un sistema que esté bien desarrollado, ya que éste debe cumplir con los requisitos que el programador haya decidido, de lo contrario, no tiene importancia ni la rapidez, ni la interfaz, ni el resto de las consideraciones que se hagan sobre él. El paso más difícil y el primero es ser

capaz de especificar de forma precisa y segura los requisitos del sistema. Este factor se subdivide en tres criterios que se muestran en [Cervera, 1997]:

- Completitud: Atributos del software que proporcionan la implementación completa de todas las funciones requeridas.
- Consistencia: Atributos del software que proporcionan uniformidad en las técnicas y notaciones de diseño e implementación.

Trazabilidad o rastreabilidad: Atributos del software que proporcionan una traza desde los requisitos a la implementación con respecto a un entorno operativo concreto.

Para garantizar la seguridad en la corrección, se emplean métodos usualmente condicionales, es decir se asegura que cada nivel sea correcto bajo el concepto de que los niveles inferiores ya han sido revisados y están correctos también.

Sucede así porque es muy difícil garantizar la corrección manejando todos los componentes y propiedades en un solo nivel.

Siguiendo esta idea, se puede decir que el software además, debe ser robusto. La robustez complementa la corrección, que tiene que ver con el comportamiento de un sistema y el cumplimiento de sus especificaciones mientras que ésta caracteriza lo que sucede fuera de la especificación.

La **robustez** tiene que ver con los casos no previstos por la especificación, esto significa que no se puede especificar lo que el sistema debería hacer porque entonces se conocerían las tareas que se quieren realizar y se volvería al terreno de la corrección. Mientras el software sea capaz de responder cuando un usuario entre datos incorrectos o en otra circunstancia que no se hubiese anticipado en los requerimientos del mismo, entonces el software es robusto. Este debe ser un requisito de fiabilidad, ya que su papel fundamental es evitar eventos catastróficos en caso de que la especificación no se contemple explícitamente, si esto sucediera el sistema debería emitir un mensaje de error y continuar su ejecución en la medida de lo posible.

Hoy en día, los ingenieros y técnicos plantean nuevas alternativas para incrementar la productividad de los desarrolladores y analistas en el desarrollo de sistemas de software, para entre otras cosas, acelerar el proceso de industrialización del mismo. Un factor importante que se ha analizado a la hora de definir la calidad de un producto es la reutilización que surge de la observación de que la mayoría de los sistemas

tenían patrones similares y en reiteradas ocasiones se inventaban soluciones a problemas que ya habían sido encontradas con anterioridad, utilizando este factor, un elemento del software reutilizable se podría aplicar en diferentes desarrollos, o sea, un producto **reparable y reutilizable**, permite que si tiene algún error los ingenieros puedan hacerse cargo con la mayor facilidad y hacerle los cambios necesarios, teniendo en cuenta que se puede construir una nueva versión del mismo.

Estas actividades se practican desde que los primeros programadores utilizaban código que habían usado anteriormente para acelerar el proceso de desarrollo. En [Soltero, 2005] se plantea que la reutilización es el procedimiento mediante el cual se produce o ayuda a producir un sistema con el uso de algún elemento procedente de un esfuerzo de desarrollo anterior. En todo el desarrollo del software aparecen varios elementos que pueden ser reutilizados.

La reutilización tiene influencia tanto en la fiabilidad como en los demás factores de la calidad ya que al resolver este problema, se escribiría menos software y sería una consecuencia positiva para el resto de los factores porque se les dedicaría más tiempo.

El desarrollo de un sistema en particular puede ser utilizado como fuente de conocimiento para la implementación de un sistema. Citando a Neighbors, uno de los iniciadores en la materia, en 1984 plantea que: “La llave de la reusabilidad software es capturada en el análisis de dominio y está centrada en reusabilidad del análisis y del diseño, no en el código”. [Soltero, 2005]

Un sistema tiene la característica de que combina diferentes elementos de software, además debe permitir la interacción con otros. En esto se basa la **compatibilidad**. La mayoría de las veces, la interacción de un sistema con otro suele tener un gran número de dificultades.

Sin embargo la clave para solucionar estos problemas está en la homogeneidad que hay en el diseño de cada producto y en acordar convenciones estándares de comunicación entre programas, por ejemplo: usar formatos de archivos estándares, estructuras de datos estándares e interfaces de usuario estándares. [Goñi, 2005]

La **portabilidad** o transportabilidad es un tema primordial tanto para los que desarrollan como para los que utilizan el software, ya que muchas de las incompatibilidades existentes entre las plataformas son injustificadas y esta es, la condición que asegura que el esfuerzo requerido para transferir un programa de un sistema a otro o de un ambiente a otro sea mínimo. La portabilidad tiene que ver con las variaciones no sólo del hardware físico sino más generalmente de la máquina hardware-software, la que realmente

programamos y que incluye el sistema operativo, el sistema de ventanas y otras herramientas fundamentales. Garantizar su seguridad es una tarea importantísima teniendo en cuenta que brinda la facilidad de transferir los productos a diferentes entornos hardware y software y un error bajo esta circunstancia seria fatal para el ingeniero que esté a cargo de dicha actividad.

La **usabilidad** expresa la facilidad con que las personas con diferentes formaciones y aptitudes pueden aprender a usar los productos y aplicarlos a la resolución de problemas. La usabilidad define los diferentes niveles de experiencia de los posibles usuarios. Este es un requisito que los diseñadores tienen como uno de sus mayores retos debido a la seguridad y facilidad de uso con que debe contar, incluyendo explicaciones y guías que sirvan para los usuarios novatos y no afecten al resto de las personas que tienen más experiencia con explicaciones monótonas. Algunos de los atributos que subdividen este requisito para entenderlo con más claridad son:

- Nivel requerido: Medido en años de experiencia con aplicaciones similares.
- Aprendizaje: Medido en horas requeridas antes de la utilización independiente.
- Capacidad de manipulación: Medida en la velocidad de trabajo después del adiestramiento y/o errores cometidos a la velocidad normal de trabajo.

Un sistema bien diseñado y construido con una estructura clara y bien pensada, es una clave segura para facilitar el uso de dicho sistema y tiende a ser más fácil de aprender y usar.

La **eficiencia** es uno de los factores más importantes que se tiene en cuenta cuando se mide la calidad del software, porque representa la capacidad que tiene el producto para exigir la menor cantidad posible de recursos hardware tales como: tiempo del procesador, espacio ocupado en memoria interna o externa o ancho de banda utilizado en los dispositivos de comunicación. Un software eficiente, requiere que el costo de su desarrollo tomando todos los recursos y el costo de su operación, debe ser tal que las organizaciones involucradas en su desarrollo y utilidad obtengan el máximo beneficio o por lo menos un beneficio aceptable en un período de tiempo establecido. Independientemente de esto es necesario que haya un equilibrio entre la eficiencia y otros factores como la reutilización ya que en un producto que no esté correcto, no es importante la eficiencia.

Una vez que el software haya sido entregado, se pasa al **mantenimiento**. Este proceso cubre toda la vida del sistema desde el momento que haya sido implantado.

Permite revisar, corregir, adaptar, mejorar y optimizar el software así como remediar los defectos del mismo. En esta etapa también influyen las modificaciones que se le pueden realizar al producto después que haya sido liberado, ya sea para corregir algún error o para hacerle cambios.

El mantenimiento de un producto permite agregar alguna nueva funcionalidad que hará mejorar la usabilidad y la aplicación del software.

Existen varios **tipos de mantenimientos** que pueden observarse en [Medina, 2007] y [Wikipedia, 2007]

1. Mantenimiento correctivo: Son las actividades que corrigen defectos y fallos detectados tanto en hardware como en software que son detectados mientras el sistema está en explotación por el usuario.
2. Mantenimiento adaptativo: Modifica el software (o hardware) de acuerdo a su entorno tecnológico. Puede aplicarse a un nuevo hardware, otro sistema de gestión de bases de datos, otro sistema operativo, etc.
3. Mantenimiento perfectivo: Actividades que se realizan para mejorar o añadir nuevas funcionalidades al sistema requeridas por el usuario. Por lo general afectan el rendimiento, flexibilidad, reusabilidad o se implementan nuevos requisitos. También se conoce como mantenimiento evolutivo.Mantenimiento preventivo: Facilitar el mantenimiento futuro del sistema (verificar precondiciones, mejorar legibilidad, etc).

En la figura se muestran los tipos de mantenimiento y el coste relativo a cada uno de ellos. Esto da idea de lo difícil que resulta mantener el software cuando no se dedican tiempo y recursos para la prevención, demostrando la singular importancia de este mantenimiento, queda claro que si se pueden corregir o prever errores a tiempo estos costes podrían disminuir gradualmente. Estas cifras son un resultado aproximado que se exponen en [Medina, 2007] de los costes que se destinan para dar mantenimiento al software.

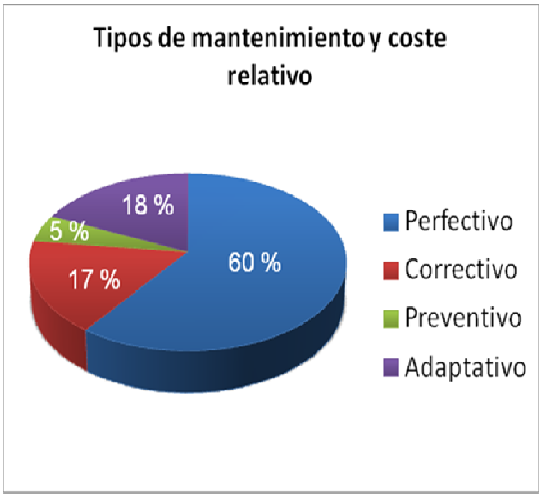

Figura 1. Mantenimiento y coste relativo del software

Cuando uno o varios ingenieros crean un producto software, utilizan un grupo de componentes que con seguridad ya han sido creado por otros ingenieros y esto lo hacen con el fin de ahorrar tiempo de desarrollo y mejorar la calidad, sin embargo en todos lo casos, se debe tener en cuenta que aunque un componente individual sea muy bueno, para que el producto tenga calidad, todos los componentes deben trabajarse en conjunto y deben estar bien combinados porque de lo contrario traería un error al programa.

El factor **fiabilidad**, es un ejemplo de lo que se ha querido expresar en el comentario anterior, es decir, un producto puede tener perfecto todos los factores que exige para que tenga calidad, pero si no es un producto fiable, entonces carece de esta característica.

La fiabilidad es un factor que se debe tratar desde el comienzo del ciclo de vida de desarrollo del producto hasta la etapa de mantenimiento del mismo ya que en cada fase puede ocurrir un error. Más adelante, en el capítulo 2 se tratará el tema de la fiabilidad en cada una de las etapas del ciclo de desarrollo del software.

1.5 TÉCNICAS UTILIZADAS EN EL DESARROLLO DEL SOFTWARE

La construcción de un software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso,

estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. El estudio y la aplicación de estas técnicas han sido efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. [Canós, 2003]

La fiabilidad del software se ha visto afectada por muchos factores durante su ciclo de vida, desde la definición del producto hasta la operación y mantenimiento del mismo. Las técnicas que se presentan a continuación están orientadas a aumentar la fiabilidad del software: [Valido, 2006]

Prevención de fallos: El objetivo de esta técnica es prevenir la introducción de errores durante el desarrollo de software. En esta categoría están incluidas todas las técnicas que se centran en el proceso de desarrollo de software: estándares, metodologías, etc. Las técnicas aquí descritas son técnicas orientadas al proceso.

Detección de fallos: Tiene el objetivo de detectar fallos una vez que el código ha sido desarrollado. Estas técnicas se centran en el producto más que en el proceso por lo tanto son técnicas orientadas al producto.

Tolerancia a fallos: Se aplica con el objetivo de proporcionar una respuesta controlada ante fallos no detectados. Estas tipo de técnicas son empleadas en el software crítico y software de alta disponibilidad.

1.5.1 Técnicas Tradicionales

Las técnicas tradicionales son las más antiguas, las que tradicionalmente el ingeniero utiliza para desarrollar un producto, ejemplo de ellas son: las encuestas, entrevistas, cuestionarios y análisis de documentación existente. Las técnicas tradicionales extraen requerimientos de las personas individuales y de los documentos. Además las pueden emplear personas sin ningún tipo de entrenamiento, sin embargo, otras técnicas como las cognitivas y contextuales demandan una especialización del personal. Por lo tanto, el esfuerzo necesario para aplicar una técnica tradicional puede ser cuantificado en términos de:

- Disponibilidad de los stakeholders (involucrados): cantidad de personas y el tiempo de disponibilidad.
- Trabajo de preparación de las sesiones.
- Entrenamiento de los ingenieros de requerimiento. [Núñez, 2005]

1.5.2 Técnicas o Métodos Formales

En el lenguaje computacional, un método es una función que realiza una acción mediante la utilización de un objeto del modelo de objetos de componentes. [Núñez, 2005]

Aquí, se verá fundamentalmente, qué son los métodos formales y cómo influyen en el desarrollo y producción de un software.

Según se plantea en [Pressman, 2005] un método es formal si posee una base matemática estable y dicha base, a su vez, está dada por un lenguaje formal de especificación. Los métodos formales permiten a los ingenieros de software especificar, desarrollar y verificar un sistema basado en computadora aplicando una notación rigurosa.

Cuando se emplean métodos formales se obtienen especificaciones representadas en lenguajes formales. Para verificarlos son aprovechadas las bases matemáticas para realizar pruebas lógicas a cada función del sistema.

El primer paso para aplicar métodos formales es la definición del invariante de datos, el estado y las operaciones. El invariante es un conjunto de condiciones que son verdaderas durante la ejecución del sistema que contiene una colección de datos; el estado por su parte es un conjunto de datos almacenados a los que el sistema accede y modifica; mientras que las operaciones son acciones que tienen lugar en un sistema y que lee o escribe datos en un estado, están asociadas a precondiciones y postcondiciones: las primeras definen las circunstancias en que una operación es válida, las segundas definen lo que sucede cuando la operación ha concluido su acción. [Pressman, 2005]

Cuando se decide aplicar métodos formales son capturados los requisitos con diferentes técnicas. Éstos son convertidos en invariantes de datos, de modo que se puedan identificar los estados y así mismo definir las operaciones.

Tienen gran importancia ya que son capaces de reducir drásticamente los errores de especificación y consecuentemente son la base del software que tiene pocos errores una vez que el usuario comienza a utilizarlo. También se denominan técnicas formales.

Estas técnicas tienen la ventaja de facilitar una mejor comprensión de los requisitos del sistema reduciendo errores y omisiones. Además proporcionan la base para un diseño elegante incluyendo la definición de pruebas y el proceso con herramientas software. [Pazos, 2000]

Los métodos formales, también tienen sus propias desventajas, por ejemplo a la hora de hacer uso de ellos, en muchos casos, no se dispone de herramientas adecuadas. Por otro lado, el uso que se le da a estos métodos es poco, debido al desconocimiento que tienen los ingenieros informáticos en el momento de aplicarlos, es por eso que los clientes sienten ciertas limitaciones cuando tienen que invertir en esta actividad ya que ellos casi nunca saben lo que realmente quieren y dejan en manos de los desarrolladores la forma de hacer el software. [Pazos, 2000]

Teniendo en cuenta que sirven como base para la verificación de programas y ayudan a detectar fallos que tal vez a simple vista no se pueden ver, en [Pazos, 2000] se planteó que se han creado diferentes metodologías basadas en las técnicas formales, y haciendo uso de ellas, es más fácil descubrir y corregir errores que presenta el software, principalmente en el diseño.

En arenas internacionales es donde se evidencia en mayor medida el uso de estos métodos porque en Cuba aún no existe una sólida industria de software capaz de promover las aplicaciones de dichos métodos a tal envergadura. Posteriormente serán abordados ejemplos que evidencian esas aplicaciones.

1.5.3 Técnicas de Evaluación del Software

No resulta tan fácil determinar la calidad, y la fiabilidad, como factor importantísimo de ella, no es la excepción. Sin embargo para ello existen técnicas de evaluación que contribuyen a integrar la fiabilidad en el proceso de desarrollo, éstas persiguen producir software sin fallos: Las técnicas de verificación y validación, que según el Estándar IEEE 729 del año 1983 propuesto en [Juristo, 2006] se definen como:

Verificación: Proceso de determinar si los productos de una cierta fase del desarrollo de software cumplen o no los requisitos establecidos durante la fase anterior.

Validación: Proceso de evaluación del software al final del proceso de desarrollo para asegurar el cumplimiento de las necesidades del cliente.

El proceso de verificación tiene que ver típicamente con errores en la transformación entre productos (de los requisitos de diseño, del diseño al código, etc.), en tanto la validación tiene que ver con errores al

malinterpretar las necesidades del cliente, es decir, si el ingeniero entendió lo que el cliente le pidió, así la única persona que puede validar el software, ya sea durante su desarrollo o una vez finalizado, es el cliente, pues éste será quién pueda detectar si fue producido adecuadamente.

De esta forma la verificación ayuda a comprobar si se ha construido el producto correctamente, mientras que la validación ayuda a comprobar si se ha construido el producto correcto.

En el capítulo 2 se hará un análisis profundo de cómo se desarrollan dichas técnicas y su influencia en las fases del ciclo de vida del software.

1.6 HERRAMIENTAS PARA EL DESARROLLO DEL SOFTWARE

Hoy en día, la mayoría de los informáticos que se dedican a la construcción de software utilizan herramientas de desarrollo basadas en tecnología orientada a objetos y que permita la reutilización del software.

Es necesario señalar que algunas herramientas no ofrecen soluciones para los problemas relacionados con sistemas o virtualmente no llevan a cabo ningún análisis de los requerimientos de la aplicación, sin embargo sí existe un conjunto de herramientas que desarrollan la cultura para muchas empresas ya que uno de sus objetivos principales es acelerar el proceso para el que han sido creadas, además de automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas, estas son las herramientas de desarrollo de software que se encargan de diseñar y construir aplicaciones y dan soporte al desarrollo e implantación de las mismas.

En la mayoría de las empresas extranjeras que producen software el trabajo se realiza con herramientas CASE (en español, Ingeniería Asistida por Computadoras) con el fin de automatizar los aspectos clave de todo lo que implica el proceso de desarrollo de un sistema.

Otras herramientas que están enmarcadas en el trabajo con las técnicas formales serán vistas en el capítulo 2.

Las herramientas de gestión de requisitos, por su parte, se basan en sistemas centralizados de gestión de bases de datos para almacenar la información correspondiente a los requisitos, que suelen consistir en párrafos de texto libre con una serie de atributos predefinidos y a los que la mayoría de herramientas permiten asociar nuevos tipos de atributos por parte del usuario.

Algunas de ellas aparecen en [Macdonald, 2005] y son expuestas las más utilizadas a continuación:

- **RequisitePro**: Es una herramienta centrada en documentos, que almacena los requisitos asociándolos a documentos (aunque también permite guardarlos directamente en la base de datos). Auxilia especialmente en el control de cambio de requisitos, con trazabilidad para especificaciones de software y pruebas. La herramienta permite el uso de Oracle sobre Unix o Windows como "back-end database" y también soporta SQL Server sobre Windows. Su principal ventaja consiste en las facilidades que brinda para que todo el equipo pueda estar al tanto del trabajo realizado con los requisitos.
- **IRqA:** Es una herramienta de ingeniería de requisitos especialmente diseñada para soportar el proceso completo de ésta. El ciclo de especificación completo incluye la captura de requisitos, análisis, especificación de sistema, validación y la organización de requerimientos es soportada por modelos estándares.

  Además se utilizan otras herramientas que por sus características pueden asumir varias funcionalidades en el proceso de desarrollo del software, en las cuales el equipo de desarrollo trabaja para la construcción del producto; las que más se destacan son:

- **Rational Rose**: Es la herramienta líder en el mundo para modelar sistemas. Es un entorno de modelado visual basado en UML[4] que permite crear, ver, modificar y manipular los componentes del modelo que se vaya desarrollando. Mejora la comunicación entre el cliente y los desarrolladores. A pesar de ser una de las herramientas más preferidas, limita su aplicación sólo a plataformas no libres.
- **Visual Paradigm**: Es una herramienta CASE que utiliza UML como lenguaje de modelado. Ofrece un diseño centrado en casos de uso y enfocado al negocio que permite una comunicación con el cliente. Además el trabajo se hace un poco más sencillo por la disponibilidad de un ambiente gráfico agradable y sencillo. [Vizcaíno, 2006] Está disponible para múltiples plataformas y su modelo y código permanece sincronizado en todo el ciclo de desarrollo del software. En la UCI se utiliza por su fácil modelado y porque posibilita el trabajo en entorno libre.

[4] UML: Lenguaje Unificado de Modelado

- **Visual Studio.NET**: Visual Studio .NET se utiliza para construir aplicaciones dirigidas a Windows, Web y dispositivos portátiles. Una de sus características más notable es el soporte de los nuevos lenguajes.NET. Visual Studio .NET Professional 2003 Special Edition permite a los desarrolladores crear rápidamente aplicaciones Web con uso intensivo de datos utilizando técnicas de Visual Basic y docenas de controles Web reutilizables e independientes del tipo de navegador. Las aplicaciones Web construidas con Visual Studio.NET y ASP.NET se benefician de un rendimiento, fiabilidad, seguridad y escalabilidad mejorados. [Desarrollo, 2004]
- **Eclipse:** Es una plataforma de herramientas universal, un entorno integrado de desarrollo abierto y extensible. Eclipse es una poderosa herramienta que permite integrar diferentes aplicaciones para construir un entorno integrado de desarrollo. Mediante Eclipse se pueden crear diversas aplicaciones como: sitios web, programas Java, C++ y Enterprise Java Beans. Una de las principales aplicaciones es JDT (Java Development Tools), herramienta para crear aplicaciones en Java. [Lead, 2007]

1.7 CONCLUSIONES PARCIALES

Construir software fiable y con calidad es una actividad siempre llena de dificultades, sin embargo, el estudio de los problemas y fallos que presentan los software, unido al avance tecnológico de los últimos años y el desempeño de cada grupo de investigación, ha dado lugar a un gran número de propuestas, métodos, metodologías y herramientas en el argot profesional informático.

Por todo lo anteriormente dicho pueden inferirse que la evolución que ha tenido la calidad la ha convertido en un aspecto relevante en el desarrollo del software y que la fiabilidad como parte de ella influye significativamente en el resultado del producto. Además se arribó a la conclusión de la superioridad que tienen las técnicas formales sobre las técnicas tradicionales siendo estas últimas las más utilizadas actualmente pero las que menos fiabilidad brindan al proceso de desarrollo del software. Este precisamente será el tema central del próximo capítulo.

CAPÍTULO II: LA FIABILIDAD DEL SOFTWARE

2.1 INTRODUCCIÓN

Cuando se tiene la idea de desarrollar un software, no importa de que tipo sea éste (software de sistema, gestión, inteligencia artificial, ingeniería, entre otros), se requiere que los analistas, diseñadores, desarrolladores y todo el personal que interviene en este proceso, apliquen características y elementos de calidad para que el producto final sea capaz de satisfacer las necesidades del usuario, que es el objetivo fundamental de sus creadores, y que a su vez este producto sea evaluado con buena calidad.

El uso eficiente de todos los factores y técnicas que intervienen en la evaluación de la calidad del software es un objetivo que aún está distante, y esto lo evidencian los reportes de fracaso y dificultades de muchos proyectos que después de haberse puesto a prueba tienen errores, necesitan cambios o sencillamente carecen de seguridad, por lo que no son productos fiables. Se dice que el éxito de un programa está en el análisis profundo que se haga en cada factor que integra la calidad y en el empleo de la metodología de desarrollo que se utilice para guiar el proceso, así como las técnicas que se empleen durante el mismo. Cada software se diseña e implementa de manera diferente porque ninguno es igual a otro, sin embargo todos requieren que al terminarlos tengan calidad.

El término fiabilidad es muy amplio en el momento de establecer una definición exacta, por eso se cree necesario aclarar que el contenido que se abordará, excluye un grupo de factores que también influyen en la fiabilidad.

Según lo anteriormente dicho en este capítulo se verán las técnicas que formalmente garantizan fiabilidad al software en todo su ciclo de vida. También podrán encontrarse las herramientas que utilizan estas técnicas para describir y verificar sistemas; además se presentan métricas que miden cuan fiable es el producto desarrollado.

2.2 FIABILIDAD DEL SOFTWARE

2.2.1 Conceptos

¿Qué es la fiabilidad?

El término fiabilidad es descrito en el Diccionario de la RAE como "probabilidad de buen funcionamiento de algo. La fiabilidad puede medirse por la frecuencia, duración y magnitud de los efectos adversos en los servicios al consumidor.

Referido al comportamiento de un sistema o dispositivo, en [Valido, 2006] se define, como la "probabilidad de que el sistema desarrolle una determinada función, bajo ciertas condiciones y durante un período de tiempo determinado."

Según se plantea en [Barba, 2004] es fiabilidad el nivel de consistencia interna o estabilidad de un instrumento cualquiera de medición (prueba) a medida que pasa el tiempo. Existen diferentes maneras de establecer un sistema fiable, por ejemplo: coeficiente de equivalencia, coeficiente de estabilidad, coeficiente de consistencia interna, fiabilidad entre evaluadores.

La fiabilidad es la probabilidad (habilidad) de un elemento de realizar una función requerida en determinadas condiciones y durante un cierto período de tiempo. Se puede decir que es "la calidad en el tiempo" [Glosario, 2000].

Según la definición informática, el término Fiabilidad, representa una característica de los sistemas informáticos por la que se mide el tiempo de funcionamiento sin fallos. En el caso del hardware, se han conseguido altísimos grados de fiabilidad, sin embargo en el software siguen existiendo bugs [5] que dificultan el buen funcionamiento de los programas. [Núñez, 2005]

La fiabilidad en un proyecto software, así como la calidad del mismo, ha sido un factor clave desde el inicio de la ingeniería del software. La seguridad no puede ser un módulo en el diseño que tiene lugar fuera de él, ni un proceso que se comprueba al finalizar el desarrollo, sino que se trata de un aspecto del proyecto que tiene que tenerse en cuenta y planificarse desde la fase de diseño, y que afectará a lo largo de todo el ciclo de vida del proyecto.

---

[5] Un bugs es un error en la codificación o en la lógica.

La fiabilidad del software se puede definir como “la probabilidad de que el software no cause un fallo del sistema durante un tiempo especificado y bajo condiciones específicas. Esta probabilidad es una función que depende de las entradas que el sistema recibe y del uso que se hace de este.” [Valido, 2006] Para darle claridad a este concepto hay que tener en cuenta que un fallo es cualquier falla de concordancia con los requisitos del software, es decir, ocurre un fallo cuando el comportamiento del sistema en tiempo de ejecución difiere de los requerimientos del cliente. Los fallos pueden ser desconcertantes o ser catastróficos. Algunos de ellos pueden ser corregidos en poco tiempo (en segundos) mientras que otros pueden tardar semanas o meses. [Casiano, 2002]

La fiabilidad de un software también está dada por la probabilidad de que un programa realice su objetivo satisfactoriamente (sin fallos) en un determinado período de tiempo y en un entorno concreto. En muchas ocasiones los fallos ocurren probabilísticamente en el tiempo de acuerdo con una tasa de intensidad de fallos. A esto se une que la corrección de un fallo puede llevar a la introducción de otros errores que, finalmente, lleven a más fallos. [Juristo, 2006]

2.2.2 Principios Básicos

Para garantizar que un software es fiable este debe carecer de factores o características que le puedan afectar, para ello se han propuesto principios básicos que fortalecen la seguridad del sistema, ellos son:

- Confidencialidad: los recursos (o funcionalidades sobre ellos) son accesibles sólo para los usuarios (o procesos) autorizados.
- Integridad: los recursos pueden ser modificables sólo por los usuarios autorizados.
- Disponibilidad: los recursos accesibles están disponibles.

Como puede verse, es evidente que la presencia de la seguridad es primordial porque afecta el sistema que se va a implantar, por tanto se debe asegurar la misma no solo en el sistema final sino en todo el proceso. La seguridad del software es una actividad de garantía de calidad del software que se centra en la identificación y evaluación de los riesgos potenciales que pueden producir un impacto negativo en el software y hacer que falle el sistema completo. Por eso la dedicación a este factor es esencial para el logro de la fiabilidad.

2.2.3 Modelos de Fiabilidad

Los modelos de fiabilidad del software generalmente se formulan en términos de procesos aleatorios y entran en dos grandes categorías: [Casiano, 2002]

- Modelos que predicen la fiabilidad como una función cronológica del tiempo (calendario).
- Modelos que predicen la fiabilidad como una función del tiempo de procesamiento transcurrido (tiempo de ejecución de CPU).

En la actualidad se han propuesto modelos mucho más sofisticados para la fiabilidad del software:

- Validez predictiva: Da la posibilidad de que el modelo prediga el comportamiento de fallos futuros basándose en los datos obtenidos de las fases de prueba y de operación.
- Capacidad: Posibilidad de que el modelo genere datos que puedan ser fácilmente aplicados a los esfuerzos de desarrollo de software industriales.
- Calidad de suposiciones: La plausibilidad de las suposiciones en las que se basan los fundamentos matemáticos del modelo cuando se llega a los límites de esas suposiciones.
- Aplicabilidad: El grado en que se puede aplicar un modelo de fiabilidad en diferentes terrenos y tipos de aplicación del software.
- Simplicidad: El grado en que el conjunto de datos que soportan el modelo es directo; el grado en que el enfoque y las matemáticas son intuitivos; el grado en que se puede automatizar el enfoque general.

2.3 PROCESO DE DESARROLLO DEL SOFTWARE

Las personas encargadas de la creación de un software deben definir un conjunto de actividades que sirvan de guía para desarrollar el producto. Estas actividades generalmente se agrupan en fases que contribuyen a obtener al menos una visión del producto final, la agrupación de estas fases es lo que se denomina **ciclo de vida del software**.

Una fase es un conjunto de actividades relacionadas con un objetivo en el desarrollo del proyecto. Se construye agrupando tareas (actividades elementales) que pueden compartir un tramo determinado del tiempo de vida de un proyecto. La agrupación temporal de tareas impone requisitos temporales correspondientes a la asignación de recursos (humanos, financieros o materiales [Ingeniería, 2007]

De manera general este ciclo incluye actividades como: toma de requisitos, análisis, diseño, desarrollo, pruebas (validación, aseguramiento de la calidad), instalación (implantación), uso y mantenimiento.

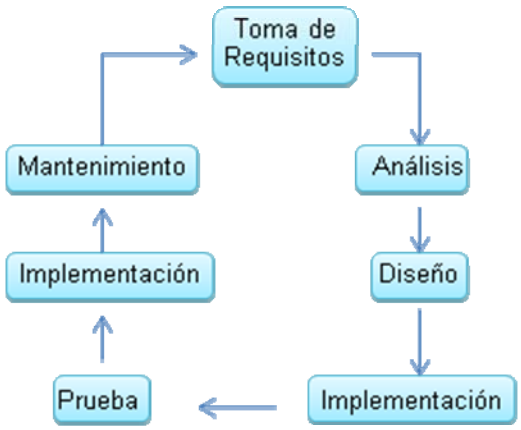

Figura 2. Ciclo de vida del software

En diferentes ocasiones se han propuesto un conjunto de modelos para el análisis y la descripción del progreso real del software, como por ejemplo: modelo lineal, en cascada, incremental, evolutivo, en espiral, o concurrente, por citar algunos. Todos estos, indudablemente contribuyen a mejorar la calidad con que terminará el producto, sin embargo hay características que marcan la diferencia entre ellos:

- El alcance del ciclo dependiendo de hasta dónde llegue el proyecto correspondiente. Un proyecto puede comprender un simple estudio de viabilidad del desarrollo de un producto, o su desarrollo completo o, llevando la cosa al extremo, toda la historia del producto con su desarrollo, fabricación, y modificaciones posteriores hasta su retirada del mercado.
- Las características (contenidos) de las fases en que dividen el ciclo. Esto puede depender del propio tema al que se refiere el proyecto (no son las mismas tareas que deben realizarse para

proyectar un avión que un puente), o de la organización (interés de reflejar en la división en fases aspectos de la división interna o externa del trabajo). [Ingeniería, 2007].

2.3.1 Modelos de Desarrollo del Software

El **modelo lineal** es ciclo es el más utilizado por ser el más sencillo de todos. Consiste en descomponer la actividad global del proyecto en fases y cada una de estas fases se realiza solo una vez. Con un ciclo lineal es fácil dividir las tareas entre equipos sucesivos, y prever los tiempos (sumando los de cada fase).

Requiere que la actividad del proyecto pueda descomponerse de manera que una fase no necesite resultados de las siguientes (realimentación), aunque pueden admitirse ciertos supuestos de realimentación correctiva. Desde el punto de vista de la gestión (para decisiones de planificación), requiere también que se sepa bien de antemano lo que va a ocurrir en cada fase antes de empezarla.

Figura 3. Modelo lineal

Por su parte el **modelo en cascada** es bastante simple y sirve de base para los demás modelos, ya que su desarrollo consiste en asignar una actividad para cada fase que servirá para complementarla y para proporcionar los requisitos de la siguiente fase, es decir que el proyecto no se diseña hasta que no haya sido analizado, o se desarrolla hasta que no haya sido diseñado, etc.

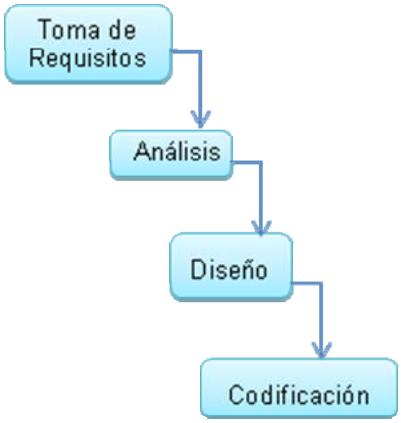

Figura 4. Modelo en cascada

El modelo en cascada se caracteriza por ser rígido, restrictivo y poco fiable, el desarrollo de las etapas se realiza de manera lineal y se deben tener bien claros los requerimientos desde el principio porque un cambio implicaría inseguridad en el proceso.

Del modelo en cascada se derivan el **incremental y evolutivo**, los que se aplican a conjuntos más pequeños del proyecto. El modelo incremental se utiliza si los subconjuntos a los que se les va aplicar, son parte del total. Trabaja de la siguiente forma, el sistema se desarrolla por partes incrementándolas y posteriormente juntándolas y así los errores producidos en un incremento son solucionados para el próximo incremento; a este paso se van añadiendo funcionalidades. El evolutivo por su parte, se aplica a versiones completas pero con menos prestaciones. [Tectimes, 2007]

El modelo **espiral**, fue desarrollado por Boehm en el año 1988 con el objeto de reunir las ventajas de los modelos de proceso software en cascada y de prototipado. [Tectimes, 2007] Cada vuelta en espiral constituye una fase del proceso.

Se puede decir que es como una generalización del modelo lineal, ya que en este último, también se descomponen las actividades en fases, pero no basta con analizarla una sola vez, es decir se realizan varias iteraciones en forma de espiral para asegurar la desaparición de incertidumbre u otras dudas que se tenga. Esto implica que su esfuerzo de desarrollo iterativo o sea que tan pronto se completa un ciclo, otro comienza. En cada iteración las especificaciones del producto se van resolviendo paulatinamente.

Las cuatro etapas fundamentales del modelo espiral pueden encontrarse en: [Vitturini, 2004]

1. Determinar objetivos, alternativas y restricciones de cada fase.
2. Evaluación y reducción de riesgos.
3. Desarrollar y probar la alternativa seleccionada.
4. Planificación.

La primera etapa de este modelo indica cuáles son los objetivos que deben trazarse para lograr un producto donde se espera que tenga buena calidad. Luego se analizan las alternativas de producción y las restricciones que esto implica. En la segunda etapa del modelo, se evalúan las alternativas definidas y se identifican los posibles riesgos que puedan surgir; también se tienen en cuenta las actividades que pueden desarrollarse para mitigar esos riesgos.

Para el tercer paso se determina primeramente un modelo apropiado (que facilite la producción) para el desarrollo. Seguidamente se comienza la construcción y prueba del próximo nivel del producto. Y finalmente en la cuarta etapa se revisan los resultados obtenidos para planificar la próxima vuelta en espiral.

En el modelo espiral, los proyectos se dividen en ciclos externos, llamados “de espiral” y se necesita de un consenso entre el cliente y los desarrolladores para finalizar el ciclo, esta es una de las diferencias que tiene con el resto de los modelos. También permite examinar y validar repetidamente los requisitos y diseños del proyecto antes de acometer nuevas fases de desarrollo. Además permite que siempre se incorporen requisitos de calidad en cada ciclo y se centra en eliminar errores y alternativas poco atractivas. Sin embargo tiene la desventaja que adaptarse fácilmente al software de paquete. [Ingeniería, 2007].

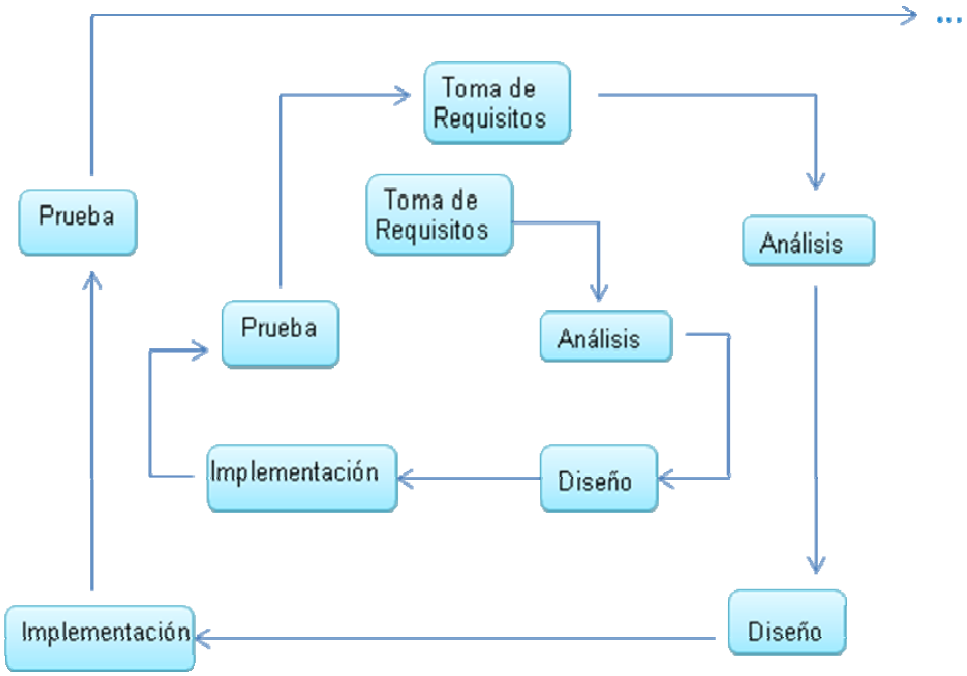

Figura 5. Modelo en espiral

Además se utiliza el **Modelo Iterativo e Incremental,** este es el que permite que las fases de análisis, diseño, desarrollo y prueba (según RUP) se retroalimenten continuamente y que se inicien lo más rápido posible. Además da la posibilidad de hacer cambios según las necesidades del usuario o utilizar nuevas herramientas o componentes que faciliten el diseño o proporcionen nuevas funcionalidades. [Ingeniería, 2007]

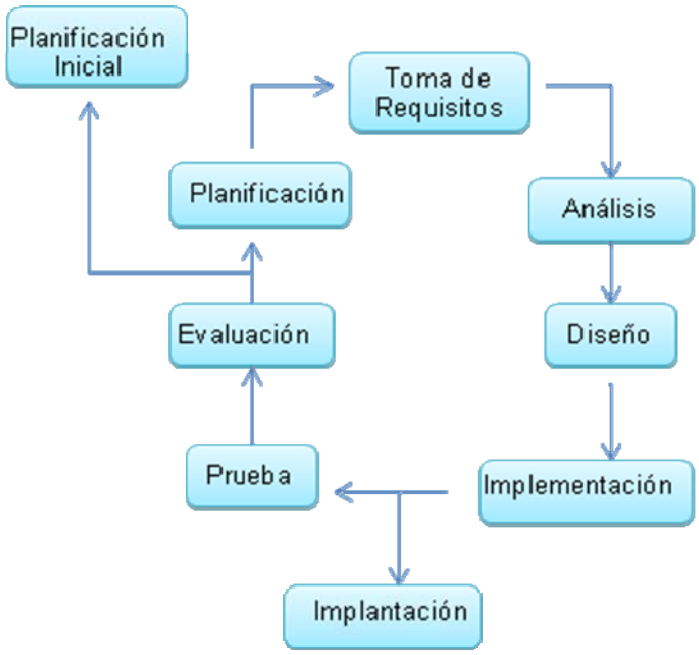

Figura 6. Modelo iterativo e incremental

La idea de utilizar cualquiera de estos modelos es obtener lo antes posible una versión funcional del software, tener en cuanta y corregir los errores que se detectaron durante la iteración anterior hasta lograr una versión fiable y con calidad.

2.3.2 Principales Metodologías utilizadas en los Proyectos Productivos de la UCI

En los proyectos productivos de la UCI se desarrollan distintos tipos de software destinados a entidades cubanas y extranjeras. Por la diversidad de la producción, se adaptan las formas de trabajo a las características particulares de los proyectos. Esto incide en que la fiabilidad del software no se trata por igual en ellos.

En las entrevistas realizadas en mayo del año 2007 a líderes de estos proyectos, se obtuvo como resultado que las metodologías más usadas son las siguientes:

RUP (Rational Unified Process): El Proceso Unificado de Desarrollo es la metodología estándar más utilizada en el ámbito informático por sus características de llevar a cabo un proceso iterativo e incremental, centrado en la arquitectura y guiado por casos de usos. Se propone asegurar la producción

de software con calidad dentro de plazos y presupuestos predecibles. Divide en cuatro fases al desarrollo del software, Inicio, Elaboración, Construcción y Transición. En cada una se generan artefactos e intervienen los roles que desarrollan los miembros de los proyectos.

Algunos proyectos donde se utiliza: Portales, FILPACON, Migración y Soporte a la Migración y en los proyectos de correo, tales como, Asistencia Postal, Oficina Multiservicios, Rastreo y Seguimiento, Correo Híbrido, ONE, CTAISC.

XP (Xtreme Programming): Como parte de las metodologías ágiles destaca por su sencillez tanto a la hora del aprendizaje como en su aplicación. Además aplica en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio. Una de las características que la distingue es que se desarrolla con pocos roles y pocos artefactos, por esto es que propone la programación en pares, que consiste en que dos desarrolladores participen en una misma estación de trabajo. Se realizan pruebas unitarias a los principales procesos para poder detectar fallos que puedan ocurrir en el futuro. [Mendoza, 2004] En el proyecto Migración y Soporte se estudian estas metodologías para ver la posibilidad de introducirlas al proyecto.

2.4 TÉCNICAS FORMALES UTILIZADAS EN EL PROCESO DE DESARROLLO DEL SOFTWARE

En la rama informática se han desarrollado muchos trabajos investigativos que han propuesto metodologías para la producción de software debido a la dificultad que esto representa.

Habiendo analizado ya el funcionamiento de los modelos más utilizados actualmente, se dará seguimiento al estudio de las técnicas formales para desarrollar software fiable, de forma que serán enmarcados sólo hasta el desarrollo de pruebas.

Trabajar con técnicas formales no significa remplazar la forma tradicional de ingeniería, sino que pueden integrarse en el proceso de desarrollo del software. Normalmente construir software cuesta mucho, si a esto se suman los costos que devienen de la aplicación de técnicas formales, esto implica duplicar los gastos de recursos y tiempo.

Tras el estudio de estas técnicas resulta evidente la importancia que tienen en la construcción de software. Sin embargo, no pueden solucionar todos los problemas que aparezcan, ya que la calidad de un producto depende de la incidencia de todos los factores analizados en el capítulo 1, aunque sin dudas, son muy provechosas sus ventajas cuando quiere lograrse un poco más de calidad de la que tradicionalmente se obtiene. [García, 2000]

Por las características de estos métodos, si no se tienen sólidos conocimientos sobre su funcionamiento, en vez de avanzar progresivamente en el proyecto, el proceso se vería afectado por no saber como usarlos, o simplemente por no seleccionar el más adecuado.

Vale destacar que los procesos de especificación y diseño están necesariamente entremezclados (ver Anexo # 5). Esto significa que antes de culminar la especificación se comienzan a realizar el diseño del futuro sistema. No existen límites entre el fin de la especificación y el inicio del diseño. Esto quiere decir que necesariamente cuando se trabajan con técnicas formales pueden perfectamente realizarse ambas disciplinas al unísono.

Otro aspecto a tener en cuenta es que muchas veces el problema no radica en el desconocimiento o falta de métodos, sino en la abundancia de ellos. Cuando se construye un producto respetable, es primordial definir un plano por el cual será guiado el proceso constructivo, es decir, seleccionar cuál sería la vía más acertada en cuanto a qué técnicas utilizar. Para esto hay que tener en cuenta aspectos como: [Palacio, 2005]

- Características de los proyectos de software gestionados.
- Visión de la organización.
- Cultura de la organización.
- Diseño y gestión del equilibrio productivo personas-procesos-tecnología.

No se trata entonces de seleccionar los mejores modelos o procesos, sino el más acorde con las necesidades que requiera el software que va a desarrollarse.

Los métodos formales están destinados a capturar la información en la especificación, arquitectura, implementación y prueba, además de obtener los requisitos indispensables para el sistema de forma consistente, haciendo del desarrollo del software un proceso automatizado. [García, 2000]

Técnicas de Descripción Formal (TDF)

Las TDF son métodos formales que posibilitan la especificación de sistemas en sus diferentes fases. [6]

Los métodos o técnicas de especificación formal, se encargan de escribir las especificaciones de los sistemas. Es decir, permiten especificar y razonar sobre el comportamiento de sistemas.

Una especificación formal es una expresión matemática que contiene la descripción de un sistema. El término TDF se utiliza para denominar cualquier método que permite definir completamente el comportamiento de un sistema (hardware o software) mediante un lenguaje con sintaxis y semántica formales. Cuando se usa un lenguaje formal para describir el modelo, se habla de una especificación formal. [García, 2000] y [Muñoz, 1998]

[García, 2000] menciona que las TDF en cada actividad del proceso de desarrollo buscan: en la fase inicial se desea obtener una especificación del sistema precisa en función de sus propiedades más relevantes. En cuanto al modelamiento se trata de describir los elementos del diseño y su interacción. Cuando se codifica, los componentes tienen que estar basados en lenguajes de especificación de modo que su comportamiento pueda ser generado por computadoras.

En [García, 2000] se plantea que hay gran variedad de técnicas de este tipo, aunque solamente presenta las siguientes clasificaciones:

- **Técnicas algebraicas**. Escriben sentencias matemáticas que expresan el efecto de las operaciones para los datos. Proporciona el razonamiento más fácil que las técnicas basadas en estados. Estas técnicas se basan en propiedades.

  Primeramente se deben definir las operaciones necesarias a través de axiomas o ecuaciones que especifican las restricciones que deben satisfacer dichas operaciones. Para esto se basan en los requisitos que fueron previamente capturados y analizados. Luego se identifican las relaciones entre operaciones. [Pressman, 2005] presenta ejemplos para este tipo de técnica.

[6] Aparecerán los términos TDF, TEF y lenguajes de especificación formales refiriéndose al mismo concepto.

Resultan útiles cuando se especifican las interfaces entre los componentes del sistema, considerando en este nivel de abstracción el sistema como un conjunto de tipos abstractos de datos. Estas técnicas presentan dificultades para la verificación formal de la completitud de las especificaciones. Ejemplo de técnicas algebraicas: Larch y OBJ.

- **Técnicas basadas en modelos matemáticos**. Basadas en teoría de conjuntos y lógica de primer orden, los más conocidos son Z, VDM y B.

  Los estados del sistema son modelados a través de los componentes del sistema, y éstos se modelan mediante conjuntos, funciones, etc. Algunas condiciones sobre los estados, se formulan por medio de predicados.

- **Técnicas basadas en lógica**: Un sistema se especifica a través de un conjunto de fórmulas de la lógica que definen relaciones y sucesos que ocurren en el tiempo. Son apropiadas cuando se va a especificar sistemas de tiempo real.

  Su principal desventaja radica en su incapacidad para expresar la estructura-arquitectura de un sistema, de ahí que se utilicen con más frecuencia en las primeras fases del diseño.

- **Técnicas basadas en álgebra de procesos**. Las técnicas citadas anteriormente modelan principalmente propiedades funcionales y comportamiento secuencial. Este hecho limita su aplicación a sistemas no concurrentes. Por su parte las basadas en álgebra de procesos modelan las interacciones entre procesos concurrentes (no secuenciales). Ejemplo: CCS, CSP y LOTOS.

  Son realmente útiles en para especificar sistemas distribuidos y concurrentes: protocolos y servicios de telecomunicaciones.

Las técnicas de descripción formal también se dividen según su funcionalidad, siendo más favorable su aplicación en algunas etapas de desarrollo que en otras:

- **No constructivas**: Tienden a usarse al principio de la especificación de requisitos.
- **Constructivas**: Se aplican para describir el diseño.

Por otra parte en [Maccoll, 1999] se determina una clasificación equivalente a la propuesta anteriormente. Ahí se presentan las técnicas formales basadas en modelos, propiedad y comportamiento, que no es más que una traslación de las anteriores, esto porque por ejemplo, las primeras se refieren a los métodos que

operan en función de modelos matemáticos; las segundas, por su parte, están estrechamente vinculadas al modelamiento de las propiedades o requisitos del sistema; los últimos, se centran en el comportamiento de los sistemas a modelar, es decir, tienen que ver con los métodos que tratan sistemas en términos de posibles secuencias de estados en vez de tipos de datos (álgebra de procesos).

Algunos ejemplos de métodos de comportamiento son: Redes de Petri, Cálculo de Sistemas de Comunicación (en inglés, Calculus of Communicating Systems: CCS), Proceso de Comunicación Secuencial (en inglés Communicating Sequential Processes: CSP). [Maccoll, 1999]

Aunque estas técnicas dan formalidad al software, no puede asegurarse que éste cumple con todos los requerimientos planteados desde el inicio del proyecto porque en ello influyen la preparación y habilidades del equipo de desarrollo y la correcta aplicación de los métodos adecuados para determinadas funciones. Una forma de comprobar qué tan bueno está quedando el sistema es precisamente comprobando las especificaciones realizadas de antemano. Esto se hace auxiliándose de otras técnicas formales como la verificación y validación de dichas especificaciones.

Como ha podido observarse el uso de estos métodos está abierto a grandes aplicaciones que incluyen mucho esfuerzo y dedicación, por eso es de suma importancia conocer a fondo las características de los mismos, antes de comenzar el trabajo con ellos. Para esto es recomendable el estudio minucioso de estas técnicas.

2.4.1 Técnicas Formales que Garantizan la Fiabilidad

Las técnicas formales que garantizan fiabilidad al software se basan en lenguajes formales para su aplicación. Estos lenguajes se han convertido en alternativa para el lenguaje natural y tienen el propósito de describir formalmente los requisitos de un sistema.

Las características que tienen estos lenguajes pueden encontrarse en: [Paredes, 2000]

- Componente semántico mínimo.
- Posibilidad de incrementar el componente semántico de acuerdo con la teoría a formalizar.
- La sintaxis produce oraciones no ambiguas.
- La importancia del rol de los números.

- Completa formalización y por esto, el potencial de la construcción computacional.

La mayor dificultad de aplicar estos lenguajes es que no favorecen la comunicación entre cliente y analista. Por otro lado, es la representación menos ambigua de los requisitos y la que más se presta a técnicas de verificación automatizadas. Estos lenguajes constituyen la forma ideal para reducir los errores e inconsistencias del software.

Estos métodos son los que serán vistos a continuación en diferentes etapas del desarrollo, sin basarse en una metodología específica, puesto que no se necesita de una para su correcto funcionamiento.

2.4.1.1 Técnicas Formales para la Especificación de Requisitos

El proceso de especificar requisitos no es más que identificar los servicios que debe proporcionar el sistema, así como las restricciones que el sistema debe cumplir para satisfacer las necesidades de los clientes y de los usuarios finales. Deben ser documentados en una serie de formatos. [Escalona, 2002] Consta de singular relevancia puesto que se definen las bases para el desarrollo del sistema a construir.

Existen varias técnicas que tradicionalmente se utilizan para realizar estas actividades; pueden ser: entrevistas, casos de usos, brainstorming (tormenta de ideas en español), lenguaje natural, revisiones, matrices de trazabilidad, entre otras, que en su momento asumen la captura, definición y validación de requisitos del software pero no de manera formal. Esta es la razón por la cual no serán analizadas en estos acápites.

En este proceso en general, pueden utilizarse lo que se conoce como **técnicas no constructivas**, o también conocidas como **orientadas a propiedades**, donde las propiedades del sistema son expresadas explícitamente (describir el sistema mediante un conjunto de requisitos que deben ser satisfechos). Tratando cada propiedad de forma separada, puede determinarse si es o no esencial para el modelamiento. Sin embargo no puede definirse si los comportamientos de las especificaciones son o no correctos. Su funcionamiento está basado en la lógica (modal y temporal) y son mayormente utilizadas en la fase inicial de la especificación de requisitos. Éstos pueden ser clasificados a su vez como axiomáticos y algebraicos: los métodos axiomáticos están basados en la lógica de predicado de primer orden. [García, 2000] y [Maccoll, 1999]

El proceso de requisitos tiene implicado las siguientes actividades:

1. Captura de requisitos
2. Definición de requisitos
3. Validación de requisitos

Comenzando por la captura de requisitos, el grupo de técnicos toma la información suministrada por usuarios y clientes. Ésta puede provenir de fuentes muy diversas: documentos, aplicaciones existentes, a través de entrevistas, etc. En base a esta información, el equipo de desarrollo define los requisitos. Finalmente con la validación de requisitos se realiza la valoración de los mismos, comprobando si existen inconsistencias, errores o si faltan algunos por definir.

Captura de Requisitos

Cuando se cuenta con un personal experimentado, esta actividad resulta más sencilla, puesto que se distinguen oportunamente los métodos a seguir, sin embargo no pueden perder de vista el cambiante mundo del negocio y siempre deben estar sujetos a factores como los mencionados en la sección 2.4.

Analizando la captura de requisitos, puede inferirse que para esta actividad no existe ninguna técnica formal porque consiste simplemente en obtener los requerimientos del software. El éxito de ello depende de la comunicación entre ingenieros y clientes, por tanto esto no puede ser automatizado.

Definición de Requisitos

Cuando son definidos explícitamente los requisitos pueden atribuírsele algunas técnicas que contribuyen a una mayor comprensión de los mismos y a acelerar el proceso de desarrollo.

- **Ontologías**: La diversidad de personas que forman parte de un proyecto software hace que sea necesario establecer un marco de terminología común. Por ejemplo, en un glosario de términos se recogen y definen los conceptos más relevantes y críticos para el sistema. [Escalona, 2002]

  En esta línea se encuentra el uso de ontologías, en las que no sólo aparecen los términos, sino también las relaciones entre ellos. Además ayudan a la especificación de los sistemas de software

posibilitando un entendimiento común entre desarrolladores y usuarios. Ejemplos donde se manifiestan estos enunciados pueden encontrarse en [Abián, 2005]:

- En campos de la inteligencia artificial (muy relacionados con la web semántica) no existen vínculos entre los conceptos: Para referirse a las mismas ideas se usan diferentes perspectivas, por lo que en ocasiones, se cometen malentendidos imperdonables. Construyendo una ontología común para esos campos, fácilmente se resolvería dicha dificultad.

Otra aplicación mucho más conocida de las ontologías, según plantea [Abián, 2005], es en el entorno de la web semántica que se han convertido en un práctico avance para la comunicación y la comparación de la información entre diferentes sistemas. Con las ontologías, se organiza la información de manera que la web pueda interpretar el significado y, por tanto, podrá buscar e integrar los datos haciendo más eficiente la búsqueda por la red[7]. La vinculación de esta técnica con la web semántica contribuye a que sus usuarios le tengan más confianza que en la web tradicional.

Gracias al conocimiento almacenado en las ontologías, las aplicaciones pueden extraer automáticamente datos de las páginas web, procesarlos y sacar conclusiones de ellos, así como tomar decisiones. [Abián, 2005] Esto es lo que muchos conocen como razonamiento automático.

Como se ha visto, las ontologías tienen un significado relevante en la definición de requisitos como en otras áreas de la informática. Vale destacar que no son consideradas en la literatura del tema como una técnica formal, pero por sus características es sólo cuestión de tiempo para que contribuyan a describir sistemas formalmente.

Validación de Requisitos

Una vez definidos los requisitos, necesitan ser validados, es decir, demostrar que los requisitos obtenidos cumplen con las necesidades que el sistema requiere. En términos tradicionales no son muchos los métodos de validación de requisitos y que por lo general se hacen manualmente. Los métodos formales que objetivamente permiten validar requisitos son:

[7] En este caso red se refiere a Internet

1. VRU
2. Larch

- **VRU (Validación de Requisitos de Usuario):** En [Sánchez, 2002] se comprobó que el método VRU permite validar requisitos de usuarios mediante prototipación automática, además de generar interfaces de usuario. Tiene la ventaja de que utiliza sólo una interfaz de usuario que posteriormente puede ser traducida y animada en diversos entornos.
- **Larch**: Es un proyecto de investigación, que según manifiestan [García, 2000] y [Larch, 2001], persigue explorar métodos, lenguajes y herramientas para facilitar el uso de las especificaciones formales. El proyecto se inició como alternativa del Instituto de Tecnología de Massachusetts (MIT por sus siglas en inglés), en el Grupo de Desarrollo de Programas Sistemáticos y en el Centro de Investigación de Sistemas Digital en Palo Alto, California.

  Son métodos algebraicos basados en propiedades que utilizan axiomas ecuacionales. Una característica que lo distingue es que cada especificación tiene componentes escritos en dos lenguajes: BISL (Lenguaje de Interfaz Larch, o en inglés, Behavioral Interface Specification Language) y LSL (Larch Shared Language). El primero se utiliza principalmente para describir las interfaces entre los componentes del programa a través de tipos abstractos de datos. El segundo es usado para describir el vocabulario matemático usado en especificaciones de pre y pos condiciones, sin importar el lenguaje. [García, 2000] y [Larch, 2001]

En esta etapa de desarrollo se utilizan técnicas formales que están orientadas a la definición y validación correcta de los requerimientos. Mejoran la seguridad de los sistemas, contribuyendo particularmente a la comprensión de los requisitos del software y a la fiabilidad del mismo, y de una forma u otra, influyendo en los otros factores de calidad. Esto quiere decir que para obtener un producto fiable lo primero que hay que garantizar es la especificación de requisitos, Si puede hacerse formalmente, mucho mejor.

Como la ingeniería de requisitos constituye un aspecto fundamental en el proceso de producción de software es de suma importancia que sus actividades se hagan lo más fiel posible. De ahí deviene la importancia del uso de las técnicas formales, puesto que las que tradicionalmente se utilizan no cubren todas las expectativas a menos que sean correctamente agrupadas.

Cuando se quiere especificar el sistema formalmente deben usarse estas técnicas que tratan las especificaciones del sistema de forma que al pasar a la siguiente etapa, sean menos los errores y sea más fácil de modelar dicho sistema.

Estándares de Requisitos

Los estándares de requisitos son usados para mejorar las prácticas de IS. Aunque no existe uno que dicte exactamente cómo especificarlos, sí contribuyen a darle más transparencia al proceso. Algunos de los más conocidos pueden verse en: [Macdonald, 2005]

- **IEEE STD- 830-1998:** Proporcionan buenas prácticas para facilitar la especificación de requisitos de software y aspectos a tener en cuenta para ello. Aunque no es obligatorio seguir sus pasos, sí se debe incluir toda la información que trae consigo. Su mayor objetivo es ayudar al equipo a entender y decidir qué es lo que realmente quiere el cliente.
- **MIL-STD 490:** Estándar Militar para las prácticas de especificación. Este estándar fue creado por el Departamento de Defensa de Estados Unidos. Está limitado a un uso militar para establecer prácticas de especificación uniformes simplemente en respuesta a necesidades de dicho departamento. Se ha mencionado como ejemplo para demostrar que los estándares no sólo son utilizados para uso civil, sino que pueden adaptarse a fines específicos. [Katz, 2003]
- **ISO/IEC 9126:** Evaluación del producto software. Es un clásico estándar de ingeniería de requisitos, creado en 1991. Tiene como objetivo principal proveer un marco para la evaluación de la calidad del software. Posibilitando la aplicación de métricas para evaluar la calidad de dichos productos, tanto externa como interna. Pueden encontrarse en [Dávila, 2006] los pasos que se siguen para la aplicación de esta norma.

Estos estándares, aunque realmente no pertenecen al grupo de los métodos formales, se consideró importante mencionarles por la ayuda que ofrecen al desarrollador y por el propósito para el que fueron creados: una mejor especificación de requisitos.

2.4.1.2 Técnicas Formales para el Modelamiento del Sistema

En el modelamiento del sistema, el principal esfuerzo es el desarrollo de la arquitectura del sistema que posteriormente será construido. Además se realizan las primeras actividades de la implementación que tienen relevancia.

Una vez llegado el momento del diseño del sistema, es recomendable el uso de métodos que formalmente establecidos, puedan darle la elegancia a la arquitectura que se quiere. Para esto se utilizan **técnicas constructivas**, que son TDF para realizar un modelado rápido del sistema final. Sin embargo, no se puede especificar las propiedades del sistema de forma explícita, lo que implica que no puedan comprobarse explícitamente algunas propiedades. [García, 2000]

- **Z**: Es una notación formal cuya pronunciación es “zed”. Fue desarrollado por miembros del Laboratorio de Computación de la Universidad de Oxford desde los años 70. Se utiliza para describir software y está basado en lógica de predicado de primer orden. Teniendo en cuenta estos fundamentos, se garantiza fiabilidad al sistema permitiendo no dar paso a ambigüedad e inconsistencias.

  Ha resultado interesante en proyectos europeos y estadounidenses, algunos como IBM CICS en el que contribuyó en 1990 a la especificación del Estándar IEEE para Binary Floating-Point Arithmetic (Aritmética Binaria del Punto Flotante). [Bowen, 2005]

  Las especificaciones en Z se modelan mediante esquemas que tienen una estructura similar a la representada en la figura 7. Un esquema es una especificación análoga a un procedimiento de un lenguaje de programación. Aquí es donde se describen el estado (declaración de variables) y el invariante representando las operaciones (predicados) de la especificación.

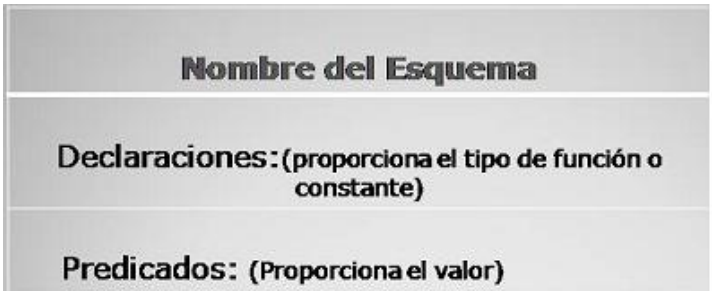

Figura 7. Representación de un esquema para Z

Para el trabajo con Z se utilizan conjuntos y lógica de primer orden. En [Pressman, 2005] se explica además otro método formal basado en objetos: Object Z. Es similar a Z pero difiere en la estructura del esquema y la inclusión de la herencia e instanciación. Una característica común es que están basadas en la noción de estados.

Algunos casos de estudios propuestos por [Bowen, 1996] evidencian la importancia requerida del uso de esta técnica. A continuación estos ejemplos:

El proyecto de Distributed Computing Software (DCS) del Laboratorio de Computación de la Universidad de Oxford, presentó un grupo de servicios de red aplicando el lenguaje formal Z en su diseño.

Incluye una particularidad como lenguaje de propósito general, y es que puede ser útil para especificar otros lenguajes como CSP, el cual es diseñado para tratar concurrencias.

- **B**: Como parte de los métodos formales derivados de Z, ha sido desarrolla igualmente por Jean-Raymond Abrial, con el fin de especificar y verificar tanto sistemas hardware y software. También incluye un grupo de herramientas que han sido aplicadas a varias áreas de la industria.

  Algunos trabajos prácticos pueden encontrarse en [B, 2007] como muestra de la Séptima Conferencia del Método B que tendrá lugar del 17 al 19 de junio de este año en Francia.

- **Larch**: Para el modelamiento del sistema tiene más influencia BISL, el cual está diseñado para lenguajes de programación específicos, usando especificaciones de pre y pos condiciones para especificar módulos de software. Además especifica interfaces y el comportamiento de los módulos en los programas. En [Leavens, 1999] se presenta una propuesta del uso de Larch (Larch/C++), el cual es un lenguaje de especificación de interfaz de comportamiento que en general es usado porque la especificación de la interfaz ayuda a la reutilización, es útil para especificar formalmente el diseño del sistema.

  Los lenguajes de interfaz Larch han sido diseñados para ser usados en lenguajes de programación como Ada, C y C++, entre otros. [Larch, 2001] Aquí están disponibles ejemplos de aplicaciones de proyectos de Larch, que no son pocos para la fecha.

- **OBJ**: [Goguen, 2005] lo presenta como una familia de lenguajes OBJ. Son lenguajes algebraicos de programación y de especificación de amplio espectro, basados en lógica de orden ecuacional, acompañado de otras como lógica ecuacional oculta o de primer orden, además proveen un poderoso sistema de módulos de programación parametrizada. Todos los lenguajes OBJ están basados fundamentalmente en sistemas lógicos. Las últimas versiones de OBJ para el año 2005, usaban álgebra de orden clasificada.

  Una de las aplicaciones de esta técnica ha sido al proyecto Tatami, el cual soporta diseño corporativo distribuido, especificación y validación de hardware y/o software, especialmente si son sistemas concurrentes distribuidos.

- **VDM (Vienna Development Method)**: El nombre de VDM deviene por alusión al laboratorio de IBM en Vienna entre los años 1973 y 1975 con el propósito de modelar sistemas computarizados, analizarlos y que esto sea útil para el diseño y el código. [Gorm, 2007]

  Entre los proyectos que ha servido VDM durante los más de treinta años de vida están: [Bjorner, 2006]

  El desarrollo del único compilador para el lenguaje CHILL y otro para ADA, convirtiéndose en el más satisfactorio.

  Algunos ejemplos interesantes se exponen en [Gorm, 2006]. Puede descargarse en este sitio la especificación de un sistema de control de municiones, el cual fue basado en regulaciones concernientes al Ministerio de Defensa del Reino Unido para la seguridad de los explosivos, por Paul Mukherjee utilizando VDM.

  Otra propuesta es la de V. Alagar, D. Muthiayen y K. Periyasamy para la especificación y diseño de un editor gráfico, que puede ser usado para crear y manipular objetos geométricos. [Gorm, 2006]

- **Estelle**: Es otra técnica de descripción formal. Aunque no exista mucha información referente a ella, pueden encontrarse algunos datos de interés. Se distribuyen un conjunto de herramientas de Estelle, que se nombran en inglés Estelle Development Toolset (EDT), que incluyen un compilador, simulador y un depurador, un generador de pruebas y otro generador de tablas de estado/evento. Ha sido aplicado a sistemas tanto Unix como Windows por esto, puede utilizarse en la esfera del software libre. [Tenney, 2000]

Algunos recursos propios de esta técnica que han tenido repercusión en la industria del software por su valor operacional son el eXperimental Estelle Compiler (XEC), que es un compilador desarrollado por la Universidad de Kaiserslautern en Alemania. Lleva implícito el nombre experimental porque se concibió sólo para experimentaciones de puesta en práctica de métodos formales. Este producto fue desarrollado para construir una plataforma de prueba, evaluación del funcionamiento y la optimización de la implementación de los métodos para Estelle. Otros ejemplos del uso de esta técnica pueden ser encontrados en [Tenney, 2000].

También existen herramientas para su aplicación, pueden ser Pet Dingo, la cual fue desarrollada por NIST, que puede encontrarse una versión en [Tenney, 2000].

- **LOTOS**: (Language of Temporal Ordering Specifications), en español, Lenguaje de Especificación de Orden Temporal. Data desde 1989, año en el cual fue introducido al mundo informático oficialmente por ISO. Entra en la clasificación de los métodos de álgebra de procesos posibilitando especificar sistemas concurrentes y distribuidos. Ha sido aplicado en varios centros universitarios y en la industria europea para la especificación de los protocolos del modelo OSI. [Belinfante, 1995]

  Resumiendo sus funcionalidades puede decirse que LOTOS expresa el comportamiento e interacciones entre procesos concurrentes. Esto se debe a que está basado fundamentalmente en CSP y CCS. Otra visión de analizar dicha técnica es la especificación de datos abstractos.

- **CSP**: Communicating Sequential Processes es un lenguaje para describir patrones de interacción en sistemas concurrentes, el cual es soportado por una teoría matemática bastante fuerte, un conjunto de herramientas de prueba y literatura extensiva. Como se ha dicho anteriormente, esta técnica entra en la clasificación de las constructivas-algebraicas.

  Ha sido aplicado en la industria del software extranjera para verificar y especificar aspectos concurrentes en varios sistemas. Las aplicaciones de la familia xCSP se han extendido a la construcción de sistemas distribuidos y paralelos propuestos en [Xcsp, 2007]. Un ejemplo: Indigo Lite, basado en CSP, integrado con un editor de código avanzado que incorpora estructuras primitivas de CSP en su código.

¿Por qué usarlos?

Los métodos han sido implementados donde el costo por fallos ha sido mayor que el costo pronosticado para el desarrollo del software como tal. En áreas internacionales la experiencia industrial indica, que cualquier costo extra en el proyecto, como capacitación del personal que usará los métodos formales, son necesarios porque son más que compensados por otros ahorros.

Existen algunos ejemplos que lamentablemente indican perfectamente la necesidad de su aplicación en proyecciones tan importantes como proyectos de aviación, actividades monetarias, equipos médicos, entre otros. Mientras más complejos sean los sistemas informáticos más se pierde por causas de fallas en dichos sistemas.

Algunos ejemplos que pueden servir para entender esto se encuentran en: [Muñoz, 1998]

En 1995, un profesor de la universidad de Lynchburg descubre un error en los procesadores Pentium de Intel. Aunque se corrigió rápidamente el error, el daño ya estaba hecho. Cerca de dos millones de procesadores defectuosos se habían distribuido alrededor del mundo. El fallo ocasional: un error de diseño en el algoritmo de división de punto flotante.

En 1996 la explosión de la nave espacial europea Ariane 5, 40 pocos segundos después de su lanzamiento dio muestra de grandes pérdidas (unos 500 millones de dólares). La comisión de expertos que investigó el desastre concluyó que la falla fue por un número fuera de rango en un pedazo de código heredado del sistema de Ariane 4. Lo irónico es que este código no tenía ninguna utilidad en el nuevo sistema.

Como contrapartida a esto se presentan aplicaciones de algunos de los métodos que mayormente son utilizados y preferidos por los desarrolladores en todo el mundo. Esta información puede encontrarse en [Maccoll, 1999]:

El proceso de ingeniería de software Cleanroom de IBM combina los métodos formales (la especificación y demostración) con las pruebas estadísticas para mejorar la calidad y la productividad.

Se hace mención del desarrollo de un sistema de información del control de tráfico aéreo combinando métodos formales. El sistema contaba con 197,000 líneas de código, con requerimientos significativos de tiempo real y de seguridad; las tasas de productividad global fueron comparables o mejores que los promedios de la industria y las tasas de error fueron significativamente mejores que los promedios de la industria.

Como se ha visto hasta el momento se han aprovechado al máximo las ventajas del uso de las técnicas formales. Esto puede demostrarse en cada ejemplo donde fueron aplicadas. En cuanto al modelamiento del sistema estas técnicas permiten que se defina un sistema formalmente establecido permitiendo detectar inconsistencias y detectar fallas en algún momento. Esto quiere decir que con el uso de los métodos formales se puede garantizar fiabilidad, utilizando herramientas adecuadas para ello. El transcurso de los años y la experiencia de los desarrolladores de software dejan claro que cuando se aplican formalismos a un sistema software se trabaja en busca de conseguir la calidad esperada por todos. De ahí la importancia de aplicarlas en las actividades desarrolladas en el diseño, porque es la base para la futura construcción del sistema.

2.4.1.3 Técnicas Formales para Construcción del Sistema

A partir de la arquitectura software definida en el modelamiento del sistema, se construyen los componentes del mismo, de forma que puedan ejecutarse en el hardware seleccionado.

Es en esta actividad donde mayor esfuerzo realizan los programadores. Por tanto son ellos los principales responsables que el resultado de su trabajo sea correcto. Para ello se basan en diferentes métodos, que no siempre son los más adecuados. Precisamente por esto es que son propuestas técnicas formales a aplicar en esta etapa también que contribuyan a dar fiabilidad al software.

Las técnicas que intervienen en esta disciplina son:

1. Interpretación Abstracta
2. Análisis Estático

- **Interpretación Abstracta:** se basa en que la semántica de un lenguaje de programación y su ejecución requiere una importante base matemática. Puede ser más o menos precisa, según el nivel de observación elegido. Es decir, si se observa sin mucho detalle, se puede realizar una abstracción de su semántica que es computable, aunque menos precisa que la original. Esto puede ser útil en cualquier etapa de desarrollo del software. [Cernuda, 2001]

Entre las ventajas de usar valores abstractos se destaca la posibilidad de asegurar que los resultados del análisis se obtienen en tiempo finito. Los resultados obtenidos describen el comportamiento del programa en algunas ejecuciones. Sin embargo tiene la desventaja que la información que se obtiene es siempre una aproximación de la que se produce en una ejecución estándar. [Gallardo, 2006]

- **Análisis Estático:** Se basa en la interpretación abstracta, dada la semántica original de un programa, derivar otra semántica aproximada y computable. Así, los ordenadores pueden evaluar en tiempo de compilación el comportamiento que un programa tendrá cuando se esté ejecutando y prever problemas que puedan aparecer.

  Sus objetivos se basan en optimizar el código generado por el compilador y validar el software que se desarrolle.

Etapas del Análisis Estático

Para realizar análisis estático se tienen en cuenta las siguientes etapas: [Sommerville, 2000]

- Análisis de flujo de control: Verifica por ciclos con puntos múltiples de salida o entrada, encuentra código errado.
- Análisis de uso de datos: Detecta variables no inicializadas, variables escritas dos veces sin una asignación intermedia, variables que se declaran pero nunca se usan, etc.
- Análisis de interfaz: Verifica la consistencia de declaraciones de rutina y de procedimiento y su uso.
- Análisis de flujo de información: Identifica las dependencias de variables de salida. No detecta anomalías él mismo pero destaca información por inspección de código o revisión.

Un caso particular es la utilización de algoritmos de análisis estáticos, donde se analiza el programa y se determinan las relaciones lineales entre sus variables. Generalmente se utilizan en la optimización de código en compiladores.

Se hace referencia también a que pudiera realizarse una “base de conocimientos” a través de un modelo de componente con todo lo que los desarrolladores del software saben respecto al comportamiento de los programas, por ejemplo: predicen que el sistema se bloqueará dado una situación. Podría verificarse si los requisitos de ciertos componentes se cumplen o no, y por tanto prever defectos de funcionamiento. Esto sería conveniente para el proyecto favoreciéndole en tiempo y calidad de producción.

En [Cernuda, 2001] se explica en detalles cómo funciona el componente ITACIO, que es un sistema de inferencia convencional basado en lógica de primer orden, para verificar si los requisitos de ciertos componentes se cumplen o no, posibilitando así la prevención de defectos de funcionamiento, entre otras funcionalidades.

Existe otro enfoque que también propone el trabajo con la interpretación abstracta cuyo objetivo es extraer información segura/correcta sobre el comportamiento dinámico de los programas sin necesidad de ejecutarlos para todas sus entradas; y hacer esto automáticamente. [Gallardo, 2006]

Uso de estas técnicas

La aplicación de estos métodos son difundidos por todo aquel que realiza software respetable. En universidades europeas y en algunos sectores de la industria de los países desarrollados, que ven una buena oportunidad en su utilización, es donde mayormente son aplicados. Esto puede demostrarse visitando varios sitios en Internet que le dan promoción que están presentes en la bibliografía de este trabajo. Lo más significativo es que ya no son pocos los centros de altos estudios que incluyen en sus planes docentes estas técnicas como una de sus asignaturas. Tal es caso de: [Upm, 2004] quienes desde el curso 2004-2005 preparan alumnos para producción de software fiable soportado con técnicas basadas en diversos cálculos lógicos y algebraicos que vienen de la teoría de tipos, orden superior, lógica difusa y temporal, álgebras y lenguajes de especificación, interpretación abstracta, cálculo basado en métodos semánticos, autómatas, cálculo funcional, deducción automática, entre otros.

Algo curioso es que buena parte de las aplicaciones prácticas de análisis estático están encaminadas a la optimización de código en compiladores, y no a la detección temprana de defectos, esto a pesar que estas técnicas llevan en explotación varios años. [Cernuda, 2001]

Pueden encontrarse ejemplos en [Nasa, 2003] donde se evidencia la utilización de la interpretación abstracta para el desarrollo de software fiable. C Global Surveyor (CGS), un programa basado en esta técnica es capaz de detectar errores automáticamente ahorrando a los programadores horas eliminando errores en el código, según expresó el investigador Arnaud Venet.

2.4.1.4 Técnicas Formales para Prueba

Una prueba es una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. [Conf7, 05-06]

El objetivo de este proceso es asegurar que el sistema implementado satisface los requisitos especificados, y las expectativas del cliente. Es aquí donde tiene mayor relevancia los procesos de verificación y validación.

Cuando se realizan las pruebas se consume mayor tiempo en el proyecto, y si no se han corregido errores del software tempranamente, las pruebas pueden demorar mucho más tiempo del imaginado y el costo del proyecto pudiera superar el presupuesto con que se dispone. Por esto deben realizarse en la medida que se va desarrollando el software en todo su ciclo de vida.

Antes de la ejecución de las pruebas, es necesario contar con el plan de prueba, de modo que se lleve a cabo todo un proceso metodológico que facilite y asegura el éxito de las mismas y no incurra en pérdidas de tiempo y costo. En el momento de la ejecución, está todo previsto, tanto los aspectos funcionales como técnicos, evitándose la improvisación.

Estas medidas, aunque no garantizan en primer plano la calidad del software, sí aportan interesantes beneficios para alcanzar los objetivos de coste, plazos y calidad final del producto terminado.

Verificación y Validación

Cuando se habla de estos términos cabría preguntarse lo siguiente: [Maccoll, 1999]

Validación: ¿Estamos construyendo el software correcto?

Verificación: ¿Estamos construyendo correctamente el software?

La respuesta a estas interrogantes determinan si realmente se está o no realizando verificación y validación al software.

Si fracasa la validación implica que no se cumplen con los contratos comprometidos a los clientes. En cambio, si fracasa la verificación significa que existen fallas en el sistema o requerimientos que no fueron previamente establecidos. [Maccoll, 1999]

Para el desarrollo de los procesos de validación y verificación se utilizan técnicas que de forma general se agrupan en dos categorías fundamentales: [Juristo, 2006]

- **Técnicas de Evaluación Estáticas**: Estas, son aquellas técnicas que buscan faltas sobre el sistema en reposo, o sea estudian los diferentes modelos que componen el sistema software y buscan posibles faltas en ellos. Estas se pueden aplicar a requisitos, código, modelos de análisis y diseño.
- **Técnicas de Evaluación Dinámicas**: Al utilizarse tienen como objetivo generar entradas al sistema para detectar fallos cuando el sistema ejecuta dichas entradas. Una vez que se detectan incongruencias entre la salida esperada y la salida real se pueden observar estos fallos. La técnica dinámica se conoce también como pruebas de software o testing y es aplicada generalmente sobre código, ya que hoy en día, es el único producto ejecutable del desarrollo.

Estos procesos persiguen comprobar la corrección de un sistema software a priori (sin ejecutarlo). [Cernuda, 2001] El uso de estos “procesos” se evidencia en todo el desarrollo del producto, de una forma u otra, cuando de técnicas formales se hable. Precisamente porque estas técnicas prometen la eliminación de errores a tiempo.

Para la verificación se proporciona una técnica basada en model checking (chequeo del modelo), utilizando la lógica temporal definida para la especificación de requisitos y el grafo de representación para realizar la verificación propiamente dicha. Está enfocado en la verificación formal de propiedades.

Las técnicas formales utilizadas en esta actividad son: [García, 2000]

1. Model Checking
2. Demostradores de Teoremas

- **Model Checking**: Es una técnica de verificación que, dado un modelo del sistema y una propiedad requerida, permite decidir automáticamente si la propiedad es satisfecha por el modelo o no.

  En otras palabras, el model checking es una técnica basada en la construcción de un modelo de estados finito del sistema y la demostración de que una propiedad dada se satisface en dicho modelo. [García, 2000] Un modelo es finito cuando las variables de estado toman un número finito de valores. [Muñoz, 1998]

  Básicamente, la demostración se realiza mediante una búsqueda exhaustiva en el espacio de estados, por lo que para que termine, este espacio de estados debería ser necesariamente finito. Desde un punto de vista técnico, las investigaciones del model checking se dirigen a la búsqueda de algoritmos y estructuras de datos más eficientes que permitan el manejo de un espacio de estados más grande.

  Las principales ventajas del model checking son: la posibilidad de automatizar totalmente las demostraciones, su rapidez, su capacidad para trabajar con especificaciones parciales, y su capacidad para proporcionar información útil aún cuando el sistema no se encuentre totalmente especificado. Estas dos últimas características hacen que el model checking sea especialmente útil en las primeras fases del desarrollo software, ya que, por una parte el sistema no está totalmente especificado y por otra, el espacio de estados sería más reducido que en las fases posteriores.

  Por contra, su principal desventaja es el problema de la explosión de estados, es decir, la explosión del espacio de búsqueda.

- **Demostradores de Teoremas**: Los demostradores de teoremas (theorem proving en inglés) se basan en expresar tanto el sistema como sus propiedades deseadas en alguna lógica matemática o mecanismos lógicos de derivación. [García, 2000] y [Muñoz, 1998] Un ejemplo es ACL2, cuyas siglas en inglés significan: “A Computational Logic for Applicative Common Lisp” desarrollado por Matt Kaufmann y Moore. [Dykstra, 2002] Esta lógica es un sistema formal que define un conjunto de axiomas y de reglas de inferencia.

  La demostración de una propiedad se realiza a través de la aplicación sucesiva de los axiomas, propiedades y reglas definidas en la lógica utilizada. El proceso finaliza cuando se obtiene una

relación, que suele ser de equivalencia o de orden, entre la especificación y la propiedad formulada.

Al contrario que la técnica de model checking, los demostradores de teoremas pueden manejar espacios de estados infinitos. Su principal desventaja reside en la dificultad de automatizar la aplicación de las reglas de inferencia para obtener un resultado sin entrar en procesos de reescritura sin fin.

Por lo general, es en especificación y modelamiento del sistema donde son cometidos la mayoría de los errores que implicarán atrasos en la construcción del software. De ahí la importancia de las técnicas de verificación, las cuales se enfocan en la detección anticipada de errores.

Uso de estas técnicas

En el desarrollo del componente ITACIO son analizadas la interpretación abstracta y el análisis estático como técnicas de verificación en [Cernuda, 2001] para evaluar el sistema, pudiendo ser aplicado en microcomponentes, diagnóstico de equipos y gestión de configuraciones a distancia.

En [Dykstra, 2002] se expone cómo son aplicados demostradores de teoremas en una herramienta que contribuye a la corrección de código. Es el caso de DESTINY, que es una plataforma que garantiza la verificación del código en Java.

Otro ejemplo es [Nasa, 2003] donde se hace mención de la herramienta PolySpace Verifier mediante la cual se efectúa la verificación automática de código para los proyectos NASA. Esta fue una opción por los agravantes acontecimientos que sucedieron producto de fallas en el software de esa institución.

Cuando finaliza la programación del software, puede decirse que ya el sistema está listo para pasar a manos de los usuarios finales. Sin embargo, la experiencia indica que no es así. Porque primeramente deben realizarse las pruebas que realmente son quienes evalúan cuán fuerte y resistente es el software desarrollado. Claro que si en todo el proceso de desarrollo del software se utilizaron métodos formales, estas actividades se sintetizan, pues resulta más rápido y fácil de medir la calidad del producto construido. Realizando pruebas mediante métodos formales a sistemas formales, se garantiza que el software final satisface con las necesidades del usuario/cliente, siendo así este el principal requerimiento para obtener la calidad de un producto.

Aunque en esta sección no fueron vistas las revisiones hechas al software, no dejan de ser seriamente importantes en esta etapa de pruebas. Como todos han de saber las verificaciones y validaciones deben realizarse iterativamente en todo el ciclo de vida del producto. Esto cobra igual importancia para las revisiones técnicas formales. Por ser un punto clave en el desarrollo del software, serán vistas en un tema a parte, y son expuestas a continuación.

2.5 HERRAMIENTAS ORIENTADAS A LAS TÉCNICAS FORMALES

Las herramientas orientadas a las técnicas formales son propiamente basadas en estas técnicas. Su funcionamiento se dirige tanto a la representación de las especificaciones como para realizar los test al software. En su mayoría no son muy conocidas porque centran su funcionamiento sólo para las técnicas que son creadas; además cada día surgen nuevas necesidades que no pueden resolver y, por tanto, quedan obsoletas en los casos que no tienen seguimiento en su desarrollo.

Herramientas para modelado con técnicas formales

Estas herramientas son construidas con el fin de modelar o especificar formalmente sistemas, de modo que brinden facilidades para comprobarlos posteriormente. Las que se presentan a continuación cumplen con estas características y algunas pueden realizar además verificación formal.

- **CZT**: Es conocido internacionalmente como Community Z Tools (Herramientas - Comunidad Z). Su nombre viene dado por el proyecto CZT que vio su aparición en septiembre del 2001. Esta gama de herramientas están previstas para editar, construcción y chequeo de especificaciones basadas en Z. En versiones recientes se incluyen funcionalidades que las anteriores carecían (incluyendo código abierto), esto demuestra el seguimiento que se le ha dado tanto al método como a las herramientas y la importancia que tienen. En [Czt, 2007] puede observarse más información respecto al tema.

En epígrafes anteriores se hacía mención de algunas herramientas que sustentan a los métodos formales: Atelier-B, B-toolkit y Model Checker. Son utilizadas para describir sistemas mediante el método B.

Atelier-B: Es una herramienta industrial que permite el uso operacional del método B para producir software de alta seguridad. Fue usado por Siemens para dar más seguridad y confianza a los usuarios del software METEOR [Atelier-B, 2007]

B-Toolkit: Abarca un grupo de herramientas integradas que proveen un desarrollo formal de los sistemas software mediante el uso del método B. Algunas de sus características son: están propuestas para interfaz Windows; administración de la configuración y la integridad de todos los archivos, incluyendo notación de máquinas abstractas, pruebas. [Piece, 2002]

NOTOS: Es una herramienta para obtener prototipos ejecutables a partir de especificaciones escritas en LOTOS. Además analiza la especificación formal de sistemas, permitiendo su verificación por medio de la simulación del comportamiento de dicho sistema. En [Gallud, 2007] se plantea su puesta en práctica obteniendo resultados satisfactorios.

Herramientas para verificación y validación

Las mismas contribuyen a garantizar la fiabilidad del software verificando y validando el trabajo realizado por todo el equipo. Una herramienta de verificación, ayuda a comprobar que el sistema especificado sea el correcto.

- **Model Checker**: Esta proporciona animación y chequeo del modelo para el método B. Brindan confianza a los usuarios tratando las especificaciones de forma segura, permitiendo detectar errores en las mismas. [Grenoble, 2007]

  Actualmente existen múltiples herramientas de model checker, tanto comerciales como de dominio público. En el ámbito académico, destacan principalmente dos: Spin y SMV (y sus variantes).

  - **Spin1**: Es una herramienta de código abierto que se utiliza para la verificación de errores de diseño en sistemas distribuidos. Fue desarrollada en los laboratorios Bell del Centro de Investigación de Ciencias de la Computación del grupo UNIX en los años 80 y se comenzó a desarrollar como producto libre desde 1991. Una de las principales optimizaciones de Spin consiste en generar dinámicamente, a partir del modelo y especificación, un programa C que contiene el algoritmo de verificación particularizado para el propio modelo. [Spinroot, 2007]

- **SMV** (Symbolic Model Verifier): Es una herramienta capaz de verificar especificaciones en modelos de sistemas concurrentes de estados finitos. Utiliza el algoritmo de model checking simbólico con OBDD. NuSMV3 es una re-implementación y extensión de SMV desarrollada en colaboración por varios centros de investigación italianos y Carnegie Mellon University. Su versión actual se distribuye con licencia LGPL.

- **JKingQA**: Es una herramienta, esta vez de análisis estático, que está destinada a facilitar y automatizar el proceso de adopción de los estándares de calidad para un departamento de calidad. Es de fácil uso. Puede dar sugerencias de cómo corregir el código. Evita que errores no detectados por el compilador, no pasen a otras fases del desarrollo. Contribuye a la reducción del coste del proyecto y a la fiabilidad del software. [Solutions, 2007]

- **C Global Surveyor (CGS)**: Visto anteriormente, es una propuesta para resolver problemas que sucedían en la NASA debido a fallas en el código del software. Tras pérdidas millonarias en el sector aeroespacial, surgió la inminente necesidad de buscar alternativas que frenaran estos acontecimientos tan deprimentes. Así fue como surgió la (CGS) implementando la técnica interpretación abstracta posibilitando la detección de errores automáticamente. [Nasa, 2003] Algo importante a destacar es que los lenguajes de programación seleccionados para las misiones en la NASA son C/C++, por tanto esta herramienta trabaja en base a la corrección de dichos lenguajes.

- **CiaoPP**: Es una herramienta basada en la interpretación abstracta. Es un pre-compilador para PROLOG que es útil para el análisis del software. Una aplicación práctica es, por ejemplo, que puede encontrar inconsistencias como incompatibilidades entre la manera en la cual se llama un predicado de una librería y su verdadero modo de uso. [Varea, 2006]

2.6 MÉTRICAS ORIENTADAS A LA FIABILIDAD DEL SOFTWARE

Para medir la fiabilidad hay que tener en cuenta la capacidad o probabilidad que tenga un sistema para ejecutarse sin presentar errores o fallos en un determinado período de tiempo. Puede medirse tanto al proceso como al producto. La importancia de medir el primero implica que si se desarrolla un buen proceso, se obtiene un buen producto. Aunque nada puede garantizar que esta actividad le de ni al proceso ni al producto la calidad que requiere.

El objetivo fundamental por el cual se mide la fiabilidad del software es tratar de predecir en que momento podría fallar el sistema, teniendo en cuenta que los fallos ocurren probabilísticamente en el tiempo según una “tasa de intensidad de fallos”. Es recomendable para ello, desde el inicio, ir comprobando el grado de fiabilidad que va teniendo el software.

Por la importancia que se otorga al correcto manejo de los requisitos es que hay que asegurar que el grado de validación de los mismos sea máximo. En la especificación de requisitos, hay que tener bien definidas cuales serán las funcionalidades del sistema. Aunque esta característica parezca cualitativa, se hace necesario que se utilicen un conjunto de métricas que garanticen que los requisitos están validados.

Asumiendo que se quiere construir un software y de él se han seleccionado las funcionalidades que especifican el sistema, se debe determinar la **especificidad de esos requisitos** de la siguiente forma:

$Q_i = N_{ui} / N_r$

Donde N_r representa la cantidad total de requisitos y N_{ui} es el número de requisitos con los que los revisores estuvieron de acuerdo en sus interpretaciones.

Una vez que los requisitos sean validados mediante el uso de las técnicas formales mencionadas anteriormente se puede calcular el **grado de validación de los mismos**:

$Q_2 = N_c / (N_c + N_{nv})$

Donde N_c es el número de requisitos validados y N_{nv} representa el número de requisitos que no se validaron.

Teniendo el grado de validación del requisito se puede decidir si pasar al modelamiento del sistema de forma fiable. Mientras mayor sea el grado de validación (Q_2) mayor será la fiabilidad. El valor de Q_2 estará entre 0 y 1 como valores límites.

La fiabilidad podría evaluarse midiendo la frecuencia y gravedad de los fallos, el tiempo medio entre fallos, la capacidad de recuperación de un fallo, la capacidad de predicción del programa y la madurez. [Gómez, 2007]

Este proceso se realiza casi siempre a través de métricas, o sea, unidades que miden la fiabilidad del sistema y se realiza contando el número de caídas operacionales.

Las métricas de fiabilidad se pueden valorar por medio de la probabilidad de fallos en demanda, es decir que el sistema no responda a la petición y además mediante la tasa de fallos, o sea, que el sistema tenga una frecuencia de comportamientos inesperados. [Sánchez, 2006]

En [Sommerville, 2000] se han definido algunas métricas que posibilitan medir el comportamiento fiable del software, por ejemplo:

El tiempo medio hasta el siguiente fallo. (TMSF)

Esta métrica está basada en el análisis que se hace con el tiempo promedio entre caídas observadas del sistema. Por ejemplo, un TMSF de 500 significa que 1 caída puede ser esperada cada 500 unidades de tiempo.

El tiempo medio de reparación del software. (TMRS).

En este caso se está hablando del tiempo promedio entre una caída de sistema y el retorno de ese sistema a ofrecer servicio.

El tiempo medio entre fallos (TMEF), que se puede calcular como: TMEF = TMRS + TMSF

La disponibilidad (D) o probabilidad de que el sistema está operando en un determinado momento se puede concebir de la siguiente manera:

D = (TMSF/TMEF) * 100%.

La probabilidad que el sistema esté disponible para su uso en un tiempo dado. Por ejemplo, una disponibilidad de 0.998 significa que en cada 1000 unidades de tiempo, el sistema es probable a estar disponible para 998 de éstas.

Capacidad de Recuperación: Es la capacidad del software de recuperarse de fallos y seguir prestando servicio, o sea, el software sigue su funcionamiento aunque aparezcan fallos en el sistema. [Hernanz, 2007] Aquí se tratan los fallos de componentes y del diseño del mismo.

Se propone en [Hernanz, 2007] la aplicación de una nueva métrica que contribuye a medir la capacidad de recuperación. Básicamente tratan de encontrar un solo valor que determine dicha recuperación mediante la siguiente fórmula:

$$\text{Capacidad de Recuperación} = \frac{r_1 m_1 + r_2 m_2 + ... + r_m m_m}{r_1 + r_2 + ... + r_m}$$

Donde cada r_i es el peso que se asigna a la métrica y m_i representa las métricas asociadas a la característica capacidad de recuperación. Los pesos asociados a cada métrica oscilarán entre 1 y 10.

Las métricas que aparecen a continuación son algunas de las mencionadas anteriormente que tienen directa relación con la capacidad de recuperación:

- Métrica tamaño del programa (m_1).
- Métrica de documentación (m_2).
- Métrica de densidad de fallos (m_3).
- Métrica de disponibilidad (m_4).
- Métrica de redundancia de funcionalidad (m_5).
- Métrica de densidad de excepciones tratadas (m_6).

Para poder entender los resultados y que sean calculables fueron convertidos los valores de estas métricas de (muy mala, mala, media, buena, muy buena a números entre 1 y 5 respectivamente).

Suficiencia de Prueba: Métrica utilizada para medir la madurez del software (para conocer cuantos de los casos de prueba necesarios están cubiertos por el plan de pruebas). La idea de dicha métrica es contar las pruebas planeadas y compararlas con el número de pruebas requeridas para obtener una cobertura adecuada.

Para ello, se utilizan las variables X, A y B

donde X = A / B

siendo A = número de casos de prueba en el plan

B = número de casos de prueba requeridos

X = nivel de madurez

0<=X

Mientras mayor sea el valor de X, mejor será la suficiencia y mayor será el nivel de madurez que tiene el software.

Eficacia de la Eliminación de Defectos (EED): Métrica que permite la eliminación de defectos.

Se define como: EED = E / (E + D)

Donde E= es el número de errores encontrados antes de la entrega del software al usuario final y D= es el número de defectos encontrados después de la entrega.

Cuando el valor de EED es 1, significa que no se han encontrado defectos en el software, pero muy pocas veces esto es así, porque en realidad D puede ser mayor que cero a medida que aumente E.

De igual forma el EED se puede analizar dentro del proyecto para evaluar la habilidad de un equipo en encontrar errores antes de que pasen a la siguiente actividad, estructura o tarea de ingeniería del software.

Para calcular el EED, en las diferentes actividades del ciclo de desarrollo (requisitos, diseño, implementación, etc.), se puede utilizar la siguiente fórmula:

$EED = E_i / (E_i + E_{i+1})$

Donde E_i = es el número de errores encontrados durante la actividad *iésima* de: ingeniería del software i, el E_{i+1} = es el número de errores encontrado durante la actividad de ingeniería del software (i + 1) que se puede seguir para llegar a errores que no se detectaron en la actividad i.

2.7 CONCLUSIONES PARCIALES

En este capítulo ha podido observarse la fiabilidad del software, mediante su estudio arribamos a conclusion de la importancia que tiene en el desarrollo del mismo. Es uno de los factores de calidad que no pueden faltar cuando se está haciendo un producto, ya que son perjudiciales las pérdidas que implica.

Una buena forma de garantizar la fiabilidad es mediante las técnicas formales aunque no hayan sido aceptadas por toda la comunidad informática en la mayor parte del mundo.

Con el estudio de las herramientas que consideramos más importantes, se comprobó que son necesarias y útiles para el trabajo y el manejo de las técnicas formales y que su uso en las diferentes etapas del software brinda al producto más eficiencia y calidad.

Un software sin fiabilidad no tiene calidad y por tanto los desarrolladores deben tener métricas para determinar el grado de fiabilidad que va teniendo el producto en la medida que se esté construyendo. La poca bibliografía referente a las técnicas formales influye en gran medida que no sean utilizadas. Casi toda la información está en inglés, para quienes no dominan el idioma, esto se convierte en una limitante.

CAPÍTULO III: PROPUESTA PARA AUMENTAR LA FIABILIDAD DEL SOFTWARE QUE SE CONSTRUYE EN LA UCI

3.1 INTRODUCCIÓN

Generalmente el desarrollo de un producto tiene resultados satisfactorios, teniendo en cuenta que se logra construir un software que resuelve el problema que el cliente planteó, sin embargo muy pocas veces se puede asegurar que dicho sistema tiene buena calidad y uno de las principales causas de este inconveniente es precisamente los fallos que se pudiera presentar el software una vez que haya sido ejecutado en algún período de tiempo, es decir que carezca de fiabilidad. Uno de los motivos podría ser el desconocimiento que existe sobre ellas y la poca información que se brinda al respecto.

En capítulos anteriores se comprobó la importancia que tiene para lograr la fiabilidad, desarrollar un producto utilizando las técnicas formales, sin embargo esta es una tarea que pocas veces se pone en práctica.

3.2 TÉCNICAS UTILIZADAS EN LA UCI

En la Universidad de las Ciencias Informáticas se ha trabajado en un grupo de proyectos que han tenido resultados positivos en la producción de software comercial, y en este momento un numeroso personal se encarga de la construcción de otros productos, según las características de estos productos, se adopta una forma de trabajo para su desarrollo.

Según la encuesta realizada (ver Anexo # 2) en la UCI no se utilizan las técnicas formales, solamente las tradicionales tales como:

- **Entrevistas**: Tradicionalmente cada vez que se quiere construir un software, los ingenieros hacen entrevistas. Esta técnica, por lo general, la ponen en práctica los líderes de proyectos cuando se reúnen con el cliente y el personal que tienen asignado. Casi siempre se convoca un encuentro donde se plantean cuales son los requisitos funcionales que debe tener el sistema y otros datos generales que se necesiten para la construcción del mismo.

- **Check List (listas de chequeo)**: En la UCI se utilizan generalmente para comprobar la interfaz de usuario, es decir, a través de esta técnica el ingeniero obtiene la información que desea sobre la interfaz del producto que está desarrollando. Son adaptadas en dependencia de las características de los proyectos. Además permite comprobar si están correctos otros factores de calidad como: usabilidad, seguridad, confiabilidad, eficiencia y portabilidad, teniendo en cuenta también aspectos de diseño. (ver Anexo # 1)

  Utilizar el check list teniendo en cuenta todos los atributos de calidad, no determina que se produzca software perfecto, pero podrían detectarse y corregirse errores que en estos momentos no salen a la luz a menos que los desarrolladores tengan experiencia.

- **Casos de Uso**: Este tipo de técnica es utilizada en casi todos los proyectos de la universidad, brindando la posibilidad de mostrar la relación que existe entre el actor y el sistema, o sea, a través de una secuencia de pasos que se describen mediante un diagrama de actividades o una descripción textual, los Casos de Uso puntualizan la secuencia de pasos que se debe tener en cuenta para desarrollar un módulo determinado del sistema.

- **Brainstorming**: También se utiliza en la captura de requisitos. Es similar a la entrevista pero incluye más personas en la reunión. En la universidad también se conoce como entrevistas.

- **Pruebas de Unidad**: Se usan cuando se está construyendo el sistema. Son pruebas que se le realizan al programa para estar seguros de que los resultados son los esperados y que carece de fallos. Con ellas se comprueba que el código esté correcto y que no se introduzcan errores si se añade una nueva funcionalidad. Es practicada por los grupos de calidad.

- **Matrices de Trazabilidad**: Según entrevistas realizadas esta última no se ha puesto en práctica pero el personal de calidad de la universidad está haciendo un estudio sobre ellas para ver si pueden aplicarse en los proyectos productivos.

- **Revisiones Técnicas Formales:** Una revisión formal se le hace al producto con el objetivo de comprobar si está funcionando fiable y correctamente. Por lo tanto las revisiones técnicas formales representan la mejor manera de garantizar que la gran mayoría de los atributos de calidad, como la fiabilidad, se cumplan.

¿Cómo se realizan las Revisiones Técnicas Formales?

Las revisiones formales como parte de las técnicas de verificación y validación juegan un papel importante en la realización de estos procesos en la universidad, debido a que constituyen una forma de comprobar en todas las fases del desarrollo que los artefactos y las actividades son realizados correctamente. Desafortunadamente estas revisiones no implican formalismos como los mencionados en este trabajo, sino que es una técnica tradicional. Además las revisiones que se hacen en la UCI no se ajustan a las normas que establece una verdadera revisión formal. Puede ver en Anexo # 4 los puntos a revisar en cada fase del desarrollo del software.

Lo que exigen las revisiones formales es:

Primeramente se seleccionan los revisores buscando siempre la forma de que no se excluyan ámbitos (futuro mantenimiento, ajuste a los estándares de la organización, usuarios y corrección, fiabilidad y calidad del producto). Luego el productor informa al jefe de proyecto que se desea hacer una revisión, posteriormente el jefe de proyecto contacta con un jefe de revisión que evalúa la disponibilidad del producto, genera copias del material del producto y las distribuye a otros revisores. Los revisores se detienen a revisar el producto no más de dos horas y se elabora una agenda para la reunión. Una revisión técnica formal se inicia con una explicación del contenido de la agenda que da el jefe de revisión; luego se discuten las dificultades que cada revisor ha encontrado y se anotan para finalmente decidir si:

- Aceptan el producto (no se le hacen modificaciones).
- Rechazan el producto (coinciden en que existen errores graves).
- Aceptan el producto modificándolo (se corrigen los errores).

Todos los revisores firman la conformidad de los resultados de la revisión y así finaliza esta actividad. Se debe tener en cuenta que cada revisión que se le haga a un software es un beneficio que se propicia. Por ejemplo si se le hace una revisión al código del producto antes de que el mismo sea terminado, se ahorra tiempo y costo, ya que los errores podrán ser detectados a tiempo, pero si la inspección se realiza desde la etapa de análisis y diseño a todos los artefactos, los ahorros serán mayores ya que es en esta etapa donde se detectan más errores. De esta forma se sugiere que las revisiones se hagan en cada etapa del desarrollo del software para que se pueda ir detectando cuales son los fallos que existen y estos sean corregidos antes de pasar a la siguiente fase.

3.3 RESULTADOS DE LA ENCUESTA

Durante el mes de mayo se realizó una encuesta (ver Anexo # 2) a un total de 35 personas de la UCI (estudiantes, profesores, líderes de proyectos y personal del departamento de calidad) con el objetivo de tener una idea de la forma y uso de los métodos utilizados en la producción de software para contribuir a la calidad de los proyectos de la universidad.

Estas personas asumían no tener mucha experiencia en la producción, la mayoría respondió que el software que se desarrollaban en la UCI no tiene óptima calidad.

El 92% de los entrevistados destacaron no haber recibido preparación suficiente como para enfrentarse a la producción que la universidad requiere. Además se señaló que no había conocimiento con respecto a las técnicas formales y esta es una de las causas por las que no se utilizan.

Por otro lado sí se dijo que en el proceso de desarrollo del software se realizan pruebas al producto con ganas de obtener calidad en el mismo aunque no siempre se logre ya que el 88 % de las encuestas señalaron que los productos que se construyen no tienen buena calidad, o al menos no tan buena, este problema entre otras cosas se debe a que el software carece de fiabilidad, precisamente porque no se tienen presente métodos que quizás podrían ser más correctos que los métodos que se utilizaron para construir el sistema.

3.4 PROPUESTAS

Propuesta de Técnicas Formales y Herramientas

Las aplicaciones de las técnicas formales, por lo general, son a sistemas grandes y complejos.

Técnicas como B, OBJ, VDM, interpretación abstracta, entre otras, ayudan a construir satisfactoriamente el software, sin embargo en la UCI no se utilizan, ya sea por el desconocimiento de las mismas o porque no se haya creído necesario. Por los resultados obtenidos en esta investigación es preciso que desde este momento se tenga en cuenta la aplicación de las técnicas formales para desarrollar software en la universidad.

La propuesta está basada fundamentalmente a técnicas de verificación y validación ya que son aplicables sin importar la complejidad que tenga producto que se desarrolle buscando en todo momento fiabilidad en el mismo, es decir son capaces de evitar que los errores detectados se extiendan a las siguientes actividades.

Para utilizar las técnicas que se presentan a continuación, no es necesario que se trabaje con una metodología específica, ya que su uso se puede adaptar a todos los proyectos de la UCI.

En la propuesta que se presentara a continuación se seleccionaron técnicas formales, herramientas y métricas que se pueden adaptar a los proyectos de la universidad ya que no se necesita una metodología específica para su uso.

Propuesta para Requisitos

Para comprobar los requisitos se propone utilizar el model checking ya que esta es una técnica que trabaja con modelos de estados finitos, lo que proporciona que las diferentes pruebas que se realizan sean más exactas.

Model Checking: Permite al verificador saber si los requisitos están correctamente seleccionados y decidir si se pasa a la siguiente disciplina de desarrollo. Esta técnica asegura rapidez a la hora de detectar los errores. Por otro lado con este modelo se logra obtener información del sistema aunque éste no esté totalmente especificado.

El modelo generalmente está expresado mediante un grafo, que consta con un conjunto de vértices y arcos, y además proposiciones que se asocian a cada nodo. Los nodos representan los estados posibles del sistema y los arcos, posibles evoluciones del sistema.

Para la aplicación del Model Checking se necesita: [Merino, 2007]

Modelar el sistema S que será analizado mediante un lenguaje de modelado cualquiera.

Especificar o establecer la propuesta P que debe satisfacer el sistema.

Verificar utilizando algún algoritmo del Model Checking (|=) si el sistema es un modelo de la propiedad: S |= P

Demostradores de teoremas: Representa una técnica que brinda al personal de calidad facilidad para conocer donde están los fallos del sistema ya que es una lógica formal que define un conjunto de axiomas y de reglas de inferencia. El uso de los demostradores de teoremas implica que se formulen propiedades y la prueba finaliza cuando ocurre una equivalencia entre la especificación y la propiedad formulada. Esta técnica es una de las más utilizadas, porque la forma de trabajo es muy ordenada.

Propuesta para Verificación de Código

Análisis Estático: El uso del análisis estático, se enmarca en la etapa de la implementación del código del software, a partir de un conjunto de simulaciones simbólicas del mismo, que permite conocer y predecir, los fallos que puede tener un programa cuando se ponga en ejecución.

Esta técnica no se utiliza en la universidad, sin embargo es muy eficiente en cuanto a la detección de errores. Para su uso, los programadores utilizan los **Analizadores Estáticos** que son herramientas de software para procesamiento de código fuente.

Ellas analizan el texto del programa y tratan de descubrir condiciones potencialmente erróneas.

En [Sommerville, 2000] se presentan los parámetros que hay que tener en cuenta para verificar el código a través de esta técnica:

Fallo de Interfaz	Se comprueba que: Todos los llamados a funciones o procedimientos tienen el número correcto de parámetros. Coinciden los tipos de parámetros reales y formales. Estén los parámetros en el orden correcto. En caso de que los componentes accedan a memoria compartida, todos tengan el mismo modelo de estructura de memoria compartida.

Fallos de administración de almacenamiento	Si una estructura ligada es modificada, deben reasignarse correctamente todas las ligas. Si se usa almacenamiento dinámico, se debe asignar espacio correctamente. Liberar espacio cuando ya no se necesite.
Fallos de administración de excepciones	Verificar que se han tomado en cuenta todas las condiciones posibles de error.

Tabla 3.1 Parámeros de verificación de código mediante análisis estático

Las Revisiones técnicas formales, según se había dicho, no entran en el grupo de las técnicas formales, sin embargo, usarlas es muy importante a la hora de la verificación y validación del software. Se utilizan en la UCI, pero se considera que no son aplicados de la forma más correcta, ya que muchas veces no se tienen en cuenta todos los atributos de calidad y no se trabaja de forma exigente en el análisis de las etapas de desarrollo. Por lo tanto, una última propuesta estaría dirigida a todos los grupos de calidad de la universidad: Extender las revisiones técnicas formales a todas las fases de desarrollo del software para asegurar que los errores de una fase no sean arrastrados a las siguientes (ver Anexo 4).

El proceso de revisión consiste en comprobar si se está desarrollando el producto correctamente, ya que permite detectar los fallos que existen en las diferentes etapas del software.

Herramienta Propuesta

La herramienta CZT facilita el uso de la técnica Z y sus derivados. En sus últimas versiones tiene aplicaciones para código abierto, esta es una de las razones por las cuales se propone, ya que nuestra pretende emigrar a software libre. Además tiene una interfaz agradable para el usuario (ver Anexo # 5) e importantes funcionalidades lo que permite el chequeo de especificaciones.

Propuesta de las Métricas de Fiabilidad del Software

En la Universidad de las Ciencias Informáticas, según las encuestas que se realizaron, no se trabaja con métricas para medir la fiabilidad que tiene el software, por lo tanto para el ingeniero no es posible conocer si cuando se concluye una fase del desarrollo el producto podría presentar fallos o estar correcto. Es válido señalar que durante las primeras actividades del proceso de desarrollo del software los errores que pueden existir son solo de requisitos, es decir, la fiabilidad aquí se demuestra teniendo 100% validados todos los requisitos funcionales del sistema. Para ello se propone utilizar la métrica de validación de requisitos:

$Q_2 = N_c / (N_c + N_{nv})$

Con ella puede verse el grado de validación que tienen los requisitos antes de pasar al modelamiento del sistema.

Durante el proceso de modelamiento, construcción y prueba del sistema, sí existe la posibilidad de que ocurran fallos, atendiendo a este criterio la propuesta es medir la fiabilidad cuantas veces sea necesario, con el fin de que, una vez que el software sea entregado cumpla con las necesidades del cliente. Para dicha actividad las métricas que se seleccionaron son:

El tiempo medio entre fallos: TMEF = TMRS + TMSF

Siendo TMSF el tiempo medio hasta el siguiente fallo y TMRS tiempo medio de reparación del software.

Utilizando esta fórmula el personal podrá conocer el tiempo (en una unidad de medidas determinada) que ocurre entre un fallo.

La disponibilidad (D): Se puede utilizar para conocer si el sistema está disponible u operando en un determinado tiempo, se calcula a través de la fórmula.

D = (TMSF/TMEF) * 100%.

Es una forma de comprobar el número de errores que presenta el software, ya que si la D es muy pequeña significa que el sistema no está funcionando correctamente y tiene fallos que hay que corregir.

La Eficacia de la Eliminación de Defectos (EED) es otra métricas que se propone para conocer los defectos que tiene el producto una vez que se haya entregado, esto permitirá saber el nivel de fiabilidad con que se terminó el software, ya que si el valor resultante de la EED es elevado quiere decir que hay bastante errores en el sistema.

La fórmula para este calculo sería:

EED = E / (E + D)

Siendo E el número de errores encontrados antes de entregar el producto y D el número de estos después de entregarlo.

Las métricas constituyen una forma de ayuda a todas aquellas personas que forman el grupo de desarrollo del software. Una métrica de fiabilidad apropiada se debe escoger para especificar la fiabilidad global del sistema. El uso de ellas sirve para prevenir y reflejar el nivel fiable del software. Por lo tanto es recomendable aplicarlas cada vez que se decida construir un producto comenzando siempre el análisis desde el principio, o sea, desde la captura de requisitos.

3.5 CONCLUSIONES PARCIALES

La aplicación de las técnicas formales para dar fiabilidad en la UCI significa implantar un nuevo modo de trabajo que contribuiría a la construcción de software fiable y daría a los ingenieros el profesionalismo que exige una institución productora de software de estos tiempos.

Cumplidos los objetivos del capítulo se arribaron a las conclusiones siguientes.

El desconocimiento de las técnicas formales en la UCI se debe fundamentalmente a que no forman parte del programa docente de la universidad ni existe una orientación relacionada con el tema; gran parte de los estudiantes limitan sus conocimientos a dicho programa.

Por tanto puede inferirse que constituirá un reto la puesta en práctica de los técnicas, herramientas y métricas propuestas porque tendrían que socializarse estos términos entre el personal de la universidad.

CONCLUSIONES

Un software que no sea fiable, no tiene calidad, ya que la fiabilidad es uno de los factores más importantes y por tanto debe analizarse desde el primer momento que comience a desarrollarse el producto.

Se ha comprobado que el uso de las técnicas formales durante todas las actividades que se realizan en el momento de construir un software, favorece la presencia de fiabilidad en el mismo, pues estos métodos son muy útiles a la hora de eliminar ambigüedades, validar y verificar que un sistema satisface sus requerimientos. Además se estudiaron un conjunto de métricas que permiten medir el grado de fiabilidad principalmente durante la construcción del sistema porque es aquí precisamente donde se comenten los errores de código que hacen fallar al software.

Se planteó una propuesta que se espera poner en práctica en la UCI, donde se abarcan técnicas formales fáciles de utilizar y que proporcionan fiabilidad al software. Las técnicas que se proponen, se enmarcan en el proceso de verificación y validación del software, ya que de una forma u otra permiten prever fallos en el sistema. Seguidamente se propusieron un conjunto de métricas para determinar los fallos que podría tener éste producto y finalmente saber si es fiable o no.

RECOMENDACIONES

Se recomienda dar seguimiento al estudio de las técnicas formales orientadas a la fiabilidad, por la importancia que ellas representan para el desarrollo del software, teniendo en cuenta que también existen técnicas tradicionales que son factibles para esta actividad.

Continuar el estudio de las técnicas formales orientadas a otros factores de la calidad del software, teniendo como punto de partida las presentadas en este trabajo, sería ideal para globalizar los conocimientos de estos temas y hacer más sólida la productividad del software cubano.

Las métricas, así como las técnicas formales orientadas a garantizar la fiabilidad del software que fueron seleccionadas, deben comenzar a ser integradas en el proceso de desarrollo del software de la universidad para garantizar la fiabilidad del software en la Infraestructura Productiva.

Se recomienda extender a otras universidades o centros de producción de software cubanos la propuesta establecida en este trabajo. De esta forma se consolidaría el desarrollo de software en todo el país orientado a búsqueda de la fiabilidad.

Se estima la posibilidad del desarrollo y publicación de un sitio web donde se expongan los temas tratados en este trabajo: los conceptos más importantes de calidad y fiabilidad, información relacionada con las técnicas formales, las herramientas, las métricas y las propuestas realizadas para aplicar en la UCI. De esta forma el personal de la universidad podrá enriquecer sus conocimientos sobre estos temas.

BIBLIOGRAFÍA

1. [Abián, 2005] Abián, Miguel Ángel. “ONTOLOGÍAS: QUÉ SON Y PARA QUÉ SIRVEN” Publicado: 9/12/2005. En sitio: “Web Semántica Hoy” Disponible en: <http://www.wshoy.sidar.org/index.php?2005/12/09/30-ontologias-que-son-y-para-que-sirven> (25/04/07)

2. [Almeraz, 2002] Almeraz, Elizabeth; Pérez-Vargas, Mariana. “Comparación práctica de los modelos de madurez SW-CMM vs CMMI” (pp. 4-5) Publicado: 19/09/2002. En sitio: Grupo Avantare. Disponible en: <http://www.avantare.com/articulos/anteriores/Un%20comparativo%20practico%20de%20los%20modelos%20de%20madurez%20v2.pdf> (12/05/07)

3. [Atelier-B, 2007] “Atelier B” En sitio: “Atelier B” Disponible en: < http://www.atelierb.societe.com/index_uk.htm> (3/05/07)

4. [B, 2007] “Tools to be presented” Publicado: 2007. En sitio: B2007. The 7th International B Conference. Disponible en: < http://lifc.univ-fcomte.fr/b2007/pages/tools.htm> (3/05/07)

5. [Barba, 2004] Barba, Dr. Robertta H. “Fiabilidad” Universidad Estatal de San José. (pp. 1) Publicado: 2004. Disponible en: <http://www.sjsu.edu/depts/it/edit221sp/wk7rel.pdf> (23/11/06)

6. [Barceló, 2006] Barceló, Miguel. “El proyecto informático de construcción de software.” UOC. Disponible en: <http://cv.uoc.es/cdocent/TN51WR4HLVHTWDZD94AZ.pdf> (12/11/06)

7. [Belinfante, 1995] Belinfante, Axel. “LOTOS” Publicado: 16/08/1995 Disponible en: < http://wwwtios.cs.utwente.nl/lotos > (19/04/07)

8. [Bjorner, 2006] Bjorner, Dines. “32 Years of VDM from Earliest Days via Adolescence to Maturity” Universidad Técnica de Dinamarca. (pp. 3) Publicado: 2006. Disponible en: <http://www.vdmportal.org/twiki/pub/Main/WebHome/bjorner-vdm-ipsj-20oct06.pdf> (3/05/07)

9. [Bowen, 1996] Bowen, Jonathan. “Formal Specification and Documentation using Z: A Case Study Approach” Capítulo 1. Edición: International Thomson Computer Press (Prensa Internacional de Computación Thomson). Publicado: 1996. Revisado: 2003. (2/05/07)

10. [Bowen, 2005] Bowen, Jonathan. “The Z notation” Centre for Applied Formal Methods. London South Bank University. Publicado: 15/09/2005. Disponible en: <http://vl.zuser.org/> (19/04 /07)

11. [Calidad, 1999] Departamento de Control de Calidad y Auditoría Informática (DCCAI) “Control de calidad en los sistemas” Dirección General de Servicios de Cómputo Académico. (pp. 5) Publicado: 29/06/1999. Disponible en: <http://sistemas.dgsca.unam.mx/publica/pdf/califormat.PDF> (24/11/06)

12. [Canós, 2003] Canós, José H; Letelier; Patricio; Penadés, Ma Carmen. “Metodologías Ágiles en el Desarrollo de Software.” (pp. 1) Universidad Politécnica de Valencia, España. Publicado: 2003. Disponible en: < http://www.willydev.net/descargas/prev/TodoAgil.Pdf > (10/02/07)

13. [Casiano, 2002] Casiano Quevedo, Julio César; García Vázquez, Israel. ”Control” Administración de Servicios de Centros de Cómputo. Unidad 6. Publicado: 12/2002. Disponible en: <http://www.itver.edu.mx/comunidad/material/serv-computo/ascc/unidad6.html> (20/11/06)

14. [Cernuda, 2001] Cernuda del Río, Agustín; Labra Gayo, Jose Emilio; Cueva Lovelle, Juan Manuel. “Verificación y validación mediante un modelo de componentes”. (pp. 1-3 y 5) Departamento de Informática de la Universidad de Oviedo. Oviedo, España. Publicado: 29-31/08/2001 en Convention Center Von Humboldt Grand Hotel, Bogotá (Colombia) y en: SISOFT-2001: Simposio Iberoamericano de Sistemas de Información e Ingeniería del Software en la Sociedad del Conocimiento. ISBN: 95897030-3-8. Disponible en: <http://www.di.uniovi.es/~cernuda/pubs/sisoft2001.pdf> (8/11/06)

15. [Cervera, 1997] Cervera Paz, Ángel; Núñez Moraleda, Bernardo M. “El modelo de McCall como aplicación de la calidad a la revisión del software de gestión empresarial” Dpto. Organización de Empresas y Dpto. Lenguajes y Sist. Informáticos, Universidad Cádiz (respectivamente). Publicado: 1997. En sitio: “Monografías.Com” Disponible en: <http://www.monografias.com/trabajos5/call/call.shtml#resu> (25/03/07)

16. [Conf7, 05-06] “Conferencia # 7. Ingeniería de Sofware II” Dartamento de Especialidad. Publicado: Curso 2005-2006. (30/01/07)

17. [Cueva, 1999] Cueva Lovelle, Juan Manuel. “Calidad del software”. Departamento de Informática, Universidad de Oviedo. España. Publicado: 21/10/1999 en Conferencia de Universidad Nacional

de la Pampa. Disponible en: <http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF> (26/03/07)

18. [Czt, 2007] “CZT” En sitio: “CZT: Community Z Tools” Publicado: 02/28/07. Disponible en: <http://czt.sourceforge.net> (30/05/07)
19. [Dávila, 2006] Dávila, Abraham; Melendez, Karin; Flores, Luis. “Determinación de los Requerimientos de Calidad del Producto Software Basados en Normas Internacionales” Pontificia Universidad Católica del Perú. Lima, Perú. Publicado: 04/2006. En sitio: Revista del IEEE de América Latina. Disponible en: <http://www.ewh.ieee.org/reg/9/etrans/vol4issue2April2006/4TLA2_6Davila.pdf> (12/06/2007)
20. [Desarrollo, 2004] “Visual Studio .NET Professional 2003 Special Edition” Publicado: 1/12/2004. En sitio: “Desarrolloweb.com” (DW) Disponible en: <http://www.desarrolloweb.com/articulos/1723.php> (20/05/2007)
21. [Dykstra, 2002] Dykstra, Josiah. “Verificación y Validación de Software con Destiny: Un enfoque paralelo a la prueba automática de teoremas” Publicado: 2002. En sitio: ACM: Crossroads Student Magazine. Disponible en: <http://www.acm.org/crossroads/espanol/xrds8-3/destiny.html> (30/04/07)
22. [Ema, 2002] Ema, Ernesto. "La evolución desde la gestión convencional a la gestión moderna del software (CMM y CMMI)" NOVOTEC (Grupo SOLUZIONA) Publicado: 2002 Disponible en: <http://www.soluziona.es/htdocs/areas/cyma/de_interes/articulos/gestion_convencional.shtml> (9/02/07)
23. [Enciclopedia, 2000] “Modal Logic (Lógica Modal).” Publicado: 02/2000. En sitio: Standford Encyclopedia of Philosophy (Enciclopedia Filosófica de Standford). Disponible en: <http://plato.stanford.edu/entries/logic-modal/> (23/04/07)
24. [Enciclopedia1, 1999] “Temporal Logic (Lógica Temporal).” Publicado: 11/1999. En sitio: Standford Encyclopedia of Philosophy (Enciclopedia Filosófica de Standford). Disponible en: <http://plato.stanford.edu/entries/logic-temporal> (23/04/07)
25. [Escalona, 2002] Escalona, María José; Koch, Nora. “Ingeniería de Requisitos en Aplicaciones para la Web. Un estudio comparativo” Departamento de Lenguajes y Sistemas Informáticos en Escuela

Técnica Superior de Ingeniería Informática. Universidad de Sevilla, España. Publicado: 12/2002. Disponible en: <http://www.pst.informatik.uni-muenchen.de/personen/kochn/ideas03-escalona-koch.pdf> (8/11/06)

26. [Fernández, 1995] Fernández Carrasco, Oscar M.; García León, Delba; Beltrán Benavides, Alfa. “Un enfoque actual sobre la calidad del software.” La Habana, Publicado: 11-12/12/1995. Disponible en: <http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm> (18/02/07)

27. [Gallardo, 2006] Gallardo Melgarejo, María del Mar; Merino Gómez, Pedro “Métodos para la Construcción de Software Fiable: Interpretación Abstracta”. Curso de Doctorado: "Ingeniería del Software e Inteligencia Artificial." Dpto. de Lenguajes y Ciencias de la Computación de la Universidad de Málaga. Disponible en: <http://www.lcc.uma.es/~gallardo/InterAbst0304.pdf> (8/11/06)

28. [Gallud, 2007] Gallud, J. A.; García Carrasco, J. M. “NOTOS: Un Generador de Prototipos para Especificaciones LOTOS” DISCA, Universidad Politécnica de Valencia y Departamento de Informática, Universidad de Castilla-La Mancha (respectivamente). Disponible en: <http://ditec.um.es/~jmgarcia/papers/NOTOS.pdf> (9/05/07)

29. [García, 2000] García Duque, Jorge. “Especificación, verificación y mantenimiento de requisitos funcionales con técnicas de descripción formal” [Tesis Doctoral] Dpto. de Tecnologías de las Comunicaciones de la Universidad de Vigo. Publicado: 2000. Disponible en: <http://idtv.det.uvigo.es/tesispdfs/tesis-jorge.pdf> (12/11/06)

30. [Glosario, 2000] “Glosario” Asesoría Técnica de Telecomunicaciones. Consultoría. Publicado: 31/08/2000. Disponible en: <http://profesionales.recol.es/ingeniero/attc/glosario.htm> (28/11/2006)

31. [Gnu, 2006] “GNU LESSER GENERAL PUBLIC LICENSE” Versión 2.1 Publicado: 2006. En sitio: GNU España. Disponible en: <http://www.es.gnu.org/modules/content/index.php?id=9> (7/06/2007)

32. [Goguen, 2005] Goguen, Joseph. “The OBJ Family” Publicado: 30/12/2005. En sitio del Departamento de Ciencia de la Computación e Ingeniería. Universidad de California. San Diego, USA. Disponible en: <http://www.cse.ucsd.edu/users/goguen/sys/obj.html> (4/05/07)

33. [González, 2006] González Rodríguez, Julia; Olsina, Luis. “Hacia La Medición de Calidad en Uso Web.” Universidad de Alicante, España. Disponible en: <http://www.dlsi.ua.es/webe01/articulos/s222.pdf> (24/11/06)

34. [González, 2007] González Arévalo, Cesar. “Calidad según Edwards Deming” Publicado: 21/03/2007. Disponible en: <http://www.gestiopolis.com/canales8/ger/calidad-por-edwards-deming.htm#autorup> (25/05/07)

35. [Goñi, 2005] Goñi, J.R; Zubizarreta, J. Iturrioz. “Ingeniería del Software. Curso 2004-2005” Departamento de Lenguajes y Sistemas Informáticos (LSI). Universidad del país Vasco. <http://siul02.si.ehu.es/~alfredo/iso/Tema1.pdf > (25/03/07)

36. [Gorm, 2006] Gorm Larsen, Peter. “The VDM-SL Examples Repository” Publicado: 10/2006. En sitio: Twiki. Disponible en: <http://www.vdmportal.org/twiki/bin/view/Main/VDMSLexamples> (3/05/07)

37. [Gorm, 2007] Gorm Larsen, Peter. “VDM Portal” En sitio: “Twiki” Publicado: 8/04/2007. Disponible en: <http://www.vdmportal.org/twiki/bin/view> (3/05/07)

38. [Grenoble, 2007] “Outils industriels / Industrial Tools” En sitio: “Grenoble B Site” (GBS) Disponible en: <http://www-lsr.imag.fr/B/Bsite-pages.html> (3/05/07)

39. [Hernanz, 2007] Hernanz Albertos, Luis Ezequiel; Minguet Melián, Jesús María. “La medida de la subcaracterística capacidad de recuperación en la norma ISO/IEC 9126 y automatización de la misma” Disponible en: <http://www.issi.uned.es/CalidadSoftware/Noticias/PonenciaCuba2005.doc> y en versión html: <http://209.85.165.104/search?q=cache:ugNX-opBsM4J:www.issi.uned.es/CalidadSoftware/Noticias/PonenciaCuba2005.doc+Capacidad+de+Recuperaci%C3%B3n+%2B+m%C3%A9trica&hl=es&ct=clnk&cd=1&gl=cu> (12/06/2007)

40. [Ingeniería, 2007] “Ingeniería del software”. Ciclos de vida. Disponible en: <http://www.telefonica.net/web2/elebo/documentos/Introd%20Ing%20Sw.PDF> (11/04/07)

41. [Instituto, 1999] “Calidad Total-Reingeniería-Normas ISO 9000.” Instituto Universitario de Tecnología Industrial. Caracas, Venezuela. Publicado: 11/1999. Disponible en: <http://html.rincondelvago.com/calidad-total_reingenieria_y_normas_iso_9000.html> (23/11/ 06)

42. [Juristo, 2006] Juristo, Natalia; Moreno, Ana M.; Vegas, Sira. “Técnicas de evaluación del software” Versión 12.0. Publicado: 17/10/2006. <http://is.ls.fi.upm.es/udis/docencia/erdsi/Documentacion_Evaluacion_7.pdf> (28/11/06)

43. [Katz, 2003] Katz, Richard. “A scientific study of the problems of digital engineering for space flight systems, with a view to their practical solution” NASA Office of Logic Design. Revisado: (17/06/2003). Disponible en: <http://www.klabs.org/DEI/References/Software/software.htm> (9/05/07)

44. [Larch, 2001] “Larch Home Page” Publicado: 11/01/2001. Disponible en: <http://nms.lcs.mit.edu/Larch> (4/05/07)

45. [Lead, 2007] “Introducción a la plataforma Eclipse”. Proyecto Pasantía LEAD Rol Plugins. Disponible en: <http://www.fing.edu.uy/inco/grupos/coal/investigacion/proyectos/lead/docs/IntroduccionEclipse.pdf html> (14/05/07)

46. [Leavens, 1999] Leavens, Gary T. “What good is it?” Departamento de Ciencias de la Computación de la Universidad del estado de Iowa. Publicado: 01/1999. Disponible en: <http://www.cs.iastate.edu/~leavens/larchc++poster/node7.html> (4/05/07)

47. [Lorés, 2006] Lorés, Jesús. "Prácticas de programación" Universidad de Lleida, España. Disponible en: <http://web.udl.es/usuaris/l4083748/pp.html> (9/02/2007)

48. [Maccoll, 1999] Maccoll, Ian; Carrington, David. "Correctitud de la Interfaz con el Usuario." Publicado: 1999. En sitio Crossroads: The ACM Student Magazine. Disponible en: <http://www.acm.org/crossroads/espanol/xrds3-3/correct.html> (28/04/07)

49. [Macdonald, 2005] Macdonald Landazuri, Bárbara A. “Definición de Perfiles en Herramientas de Gestión de Requisitos” Programa de Doctorado. Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del Software. Facultad de Informática de la Universidad Politécnica de Madrid, España. Publicado: 09/2005. Disponible en: <http://is.ls.fi.upm.es/doctorado/Trabajos20042005/Mcdonald.pdf> (8/05/07)

50. [Medina, 2007] Medina Patón, Dr. Eduardo F. “Mantenimiento del Software” Tipos de Mantenimiento y Coste Relativo. (pp. 5) Materiales para 4to Curso de Ingeniería del Software I:

2006-2007. Departamento de Tecnologías y Sistemas de Información de la Universidad de Castilla-La Mancha. En sitio: Alarcos. Disponible en: <http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema16.pdf> (20/04/07)

51. [Mendoza, 2004] Mendoza Sánchez, María A. Metodologías De Desarrollo De Software. TeamSoft Perú S.A.C. Publicado: 7/06/2004. Disponible en: <http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html> (17//05/07)

52. [Merino, 2007] Merino Gómez, Pedro; Gallardo Melgarejo, María del Mar. “Métodos para la Construcción de Software Fiable” [Curso de Doctorado: "Ingeniería del Software e Inteligencia Artificial"] Disponible en: <http://www.lcc.uma.es/~gallardo/modelchecking1.pdf> (28/05/07)

53. [Miranda, 2006] Miranda, Lic. Sandor Luis. “Sistema de Gestión de La Calidad” Publicado: 6/10/2006. Disponible en: <http://www.gestiopolis.com/canales7/ger/sistemas de gestión de la calidad.htm> (24/11/06)

54. [Muñoz, 1998] Muñoz, César. “Métodos Formales, por ejemplo” Computer Science Laboratory, SRI International. Publicado: 1998. En sitio: NIA: National Institute of Aerospace. Disponible en: <http://www.csl.sri.com/users/munoz/Papers/mfhtml/mfhtml.html > (28/04/07)

55. [Nasa, 2003] “La computadora lo puede verificar” Publicado: 2003. En sitio: National Aeoronautics and Space Administration (NASA). Disponible en: <http://www.nasa.gov/centers/ames/spanish/research/exploringtheuniverse/exploringtheuniverse-computercheck.html> (30/04/07)

56. [Normas, 2007] “¿Qué es ISO?” En sitio: Normas 9000. Disponible en: <http://www.normas9000.com/intro normas 9000.aspx> (10/02/07)

57. [Núñez, 2005] Núñez Camallea, Noel Luis; Coutin Ronald. “Diccionario de Informática” Editorial Científico -Técnica. Publicado: 2005. (12/06)

58. [Palacio, 2005] Palacio, Juan. “Gestión y modelos para la eficiencia en empresas de desarrollo software” En sitio: BAQUIA: Centro de Conocimiento. Publicado: 12/05/2005. Disponible en: <http://www.baquia.com/noticias.php?id=9651> (8/11/06)

59. [Paredes, 2000] Paredes Hernández, Cutberto Uriel. “Procesamiento Computacional del Lenguaje Natural” Tamaulipas, México. Publicado: 30/09/2000. Disponible en: <es.geocities.com/plninformacion/doc/procesamiento_del_lenguaje_natural.doc> (25/04/07)

60. [Pavón, 2004] Pavón Mestras, Juan. "Garantía de calidad del software. Ingeniería del Software de Gestión I" Dep. Sistemas Informáticos y Programación en Facultad de Informática de Universidad Complutense Madrid. Publicado: 2004. Disponible en: <http://www.fdi.ucm.es/profesor/jpavon/is1/10Calidad.pdf> (8/02/07)

61. [Pazos, 2000] Pazos Arias, José Juan. "Especificación Formal. Técnicas para la especificación no ambigua del software" Ingeniería del software de comunicaciones. Publicado: 21/02/2000. Disponible en: <http://www-gris.det.uvigo.es/~jose/doctorado/fdt/sld001.htm> (8/02/07)

62. [Piece, 2002] Piece Harwell, King. “The B-Toolkit” Publicado: 02/2002 en Reino Unido. En sitio: B-Core. Disponible en: <http://www.b-core.com/btoolkit.html> (3/05/07)

63. [Portal, 2006] “Norma de Calidad. Gestión de La Calidad o Excelencia” En sitio: Buscar Portal. Disponible en: <http://www.buscarportal.com/articulos/iso 9001 gestión calidad.html> (23/11/06)

64. [Portal1, 2006] “Sistemas de Gestión de la Calidad, Requisitos” En sitio: Buscar Portal. Disponible en: <http://www.buscarportal.com/articulos/iso_9001_2000_gestión_calidad.html> (30/01/07)

65. [Pressman, 2005] Pressman, Roger S. “Ingeniería del Software. Un enfoque práctico.” Editorial: “Félix Varela.” La Habana, 2005. (1/12/06)

66. [Sánchez, 2002] Sánchez, Juan; Belenguer, Jorge; B Belenguer, Pablo; Pascual, David. “VRU: Un método para validar requisitos y generar interfaces de usuario multiplataforma” Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, España. En sitio: ”OO Method” Publicado: 2002. Disponible en: <http://oomethod.dsic.upv.es/files/InConferenceArticle/2002_WER.pdf> (26/04/07)

67. [Soltero, 2005] Soltero Domingo, Francisco J.; Bodas Sagi, Diego J.; Pozo Llorente, Valentín. “Reutilización en el Dominio del Análisis Software” Ingeniería Técnica de Informática de Sistemas. Revista “Enlaces” del Centro de Estudios Superiores (CES) Felipe II. ISSN: 1695-8543. Publicado: 2005. Disponible en: <http://www.cesfelipesegundo.com/revista/articulos2005/Informatica1.pdf> (26/03/07)

68. [Solutions, 2007] “Herramientas para el entorno de pruebas” En sitio: Aplications LifeCycle Solutions (ALS). Copyright: 2007. Disponible en: <http://www.als-es.com/home.php?location=herramientas/entorno-pruebas> (9/05/07)

69. [Sommerville, 2000] Sommerville, Ian. "Verificación y Validación" (Prentice Hall). Ingeniería de Software, Capítulo 19, 6ta. Edición, Disponible en: <http://ccc.inaoep.mx/~edominguez/IngSoftIcap19.pdf> (29/05/07)

- Sommerville, Ian. "Especificación de Sistemas Críticos" (Prentice Hall). Ingeniería de Software, Capítulo 17, 6ta. Edición, Disponible en: <http://ccc.inaoep.mx/~grodrig/Descargas/IngSoftIcap17.pdf> (29/05/07)

70. [Spinroot, 2007] “On-The-Fly, LTL Model Checking with SPIN” En sitio: SPIN. Modificado: 3/06/2007. <http://spinroot.com/spin/whatispin.html> (31/05/07)

71. [Tectimes, 2007] “IMPLEMENTACIÓN Y DEBUGGING” *Capítulo 1: Ciclo de vida del Software*. <www.tectimes.com/lbr/Graphs/revistas/lpcu097/capitulogratis.pdf> (11/06/07)

72. [Tenney, 2000] Tenney, Richard L. “Estelle Information” Publicado: 25/04/2000. Disponible en: <http://www.estelle.org> (19/04/07)

73. [Upm, 2004] “Máster Europeo en Computación Lógica” Facultad de Informática de la Universidad Politécnica de Madrid (UPM). Publicación: 2004. Disponible en: <http://www.clip.dia.fi.upm.es/mastercl/indice.html> (30/04/07)

74. [Valido, 2006] Valido Cabrera, Eduardo. “Software reliability methods” Universidad Politécnica de Madrid. Publicado: 08/2006. Disponible en: <http://www.is.ls.fi.upm.es/doctorado/Trabajos20052006/Valido.pdf> y <http://is.ls.fi.upm.es/doctorado/Trabajos20052006/Valido_English.pdf> (23/11/06)

75. [Varea, 2006] Varea, Mauricio. “Especialización y Análisis Avanzados de Sistemas Pervasivos” Publicado: 4/05/2006. Disponible en: <http://users.ecs.soton.ac.uk/~mv/research/asap.php?lang=es> (9/05/07)

76. [Vitturini, 2004] Vitturini, Lic. María Mercedes. ”Elementos de Bases de Datos” Dpto. Ciencias e Ingeniería de la Computación Universidad Nacional del Sur. Publicado: 2004. Disponible en:

<http://cs.uns.edu.ar/~gis/ebd/Archivos/Clases/EBD%20-%20Clase%2002%202004%20BN.pdf> (11/04/07)

77. [Vizcaíno, 2006] Vizcaíno, Aurora; García, Felix Óscar; Caballero, Ismael. “Una Herramienta CASE para ADOO: Visual Paradigm. Análisis y Diseño Orientado a Objetos” Prácticas Ingeniería del Software 3º. Universidad de Castilla-La Mancha. Publicado: 20006. Disponible en: <http://alarcos.inf-cr.uclm.es/per/fgarcia/isoftware/doc/LabTr1_VP.pdf> (9/06/2007)

78. [Wikipedia, 2007] “Mantenimiento de software” Modificado: 27/05/2007. En sitio: Wikipedia. Disponible en: <http://es.wikipedia.org/wiki/Mantenimiento_de_Software> (28/05/07)

79. [Xcsp, 2007] “Applications for xCSP” En sitio: “xCSP” Disponible en: <http://www.quickstone.com/xcsp/applications> y en <http://www.quickstone.com/xcsp/products/> (5/05/07)

GLOSARIO

1. Axioma: Es una proposición o evidencia que no es susceptible de demostración. Sirven de base para los modelos matemáticos y son “verdades evidentes” sobre la cual descansa conocimiento humano. Los axiomas se refieren a los principios que se aceptan en todas las ciencias; en las matemáticas se usan para poder realizar operaciones lógicas.
2. Datos Abstractos: Son modelos matemáticos compuestos por una colección de operaciones definidas sobre un conjunto de datos para el modelo. La abstracción de datos consiste en ocultar las características de un objeto y obviarlas, de manera que solamente se utiliza el nombre del objeto en el programa.
3. Licencia LGPL: Lesser General Public License (Licencia Pública General Menor). Esta licencia se aplica a cualquier programa o trabajo que contenga una nota puesta por el propietario de los derechos del trabajo estableciendo que su trabajo puede ser distribuido bajo los términos de la GPL (General Public License). Se aplica a algunos paquetes de software diseñados. están diseñadas para asegurar que usted tiene la libertad de distribuir copias de software libre (y cobrar por este servicio si lo desea); que recibe el código fuente o que puede obtenerlo si lo quiere; que puede modificar el software y usar partes de él en nuevos programas libres; y de que ha sido informado de que puede hacer estas cosas. [Gnu, 2006]
4. Lógica Modal: La lógica modal es el estudio del comportamiento deductivo de las expresiones 'necesariamente' y 'posiblemente'. Sin embargo se utiliza para cubrir una familia de lógicas con reglas similares y de una variedad de diversos símbolos. En resumen se centra en el estudio de la noción de necesidad lógica, que está basado en reglas identificadas a través de diferentes símbolos. [Enciclopedia, 2000]
5. Lógica Temporal: El término lógica temporal se ha utilizado para cubrir los acercamientos a la representación de la información temporal dentro de un marco lógico, y también para referirse específicamente al tipo de la lógica modal de acercamiento, antes conocido bajo el nombre de *tense logic* (lógica de tiempo). Las aplicaciones de la lógica temporal incluyen su uso como un formalismo para clarificar ediciones filosóficas sobre el tiempo, para definir la semántica de expresiones temporales en lenguaje natural, como un lenguaje para codificar conocimiento temporal en inteligencia

artificial, y como herramienta para manejar los aspectos temporales de la ejecución de los programas de computadora. [Enciclopedia1, 1999]

6. OSI: (OSI, Open System Interconnection) fue lanzado en 1984, siendo el modelo de red descriptivo creado por ISO. Representa el modelo de referencia de Interconexión de Sistemas Abiertos. Está dividido en capas; su estructura permite la comunicación entre ellas y asegura que los datos lleguen correctamente al otro lado de la comunicación.
7. Brainstorming: Es una técnica tradicional utilizada para la captura de requisitos, también conocida como tormenta de ideas por su significado en español. Está basada en reuniones de grupos donde los participantes muestran sus ideas de forma libre. Consiste en la acumulación de ideas y/o información sin evaluar las mismas, ofreciendo sólo una visión general del sistema sin entrar en detalles específicos. [Escalona, 2002]
8. Herramienta CASE: Las Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por computadora) son diversas aplicaciones informáticas y herramientas semi-automatizadas y automatizadas destinadas a conseguir la generación automática de programas desde una especificación a nivel de diseño. Estas herramientas resultan de mucha ayuda en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, calculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.
9. Base de conocimiento: Una base de conocimiento almacena información a cerca de un dominio en particular. Esta información contiene representaciones simbólicas de reglas y la experiencia práctica del experto en una forma que el mecanismo de inferencia puede procesar. [Núñez, 2005]