Universidad las Ciencias Informáticas

FACULTAD 10

Título: "Técnicas Formales para la evaluación de factores externos de calidad del software"

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

Autores: Yadira Morales Alamo
Maikel Lazaro Jimenez Moreira

Tutores: MSc. David Leyva Leyva
Ing. Susel Cañete Pollán
MSc. Daymy Tamayo Avila

Ciudad de La Habana
Julio 2008

"La responsabilidad nuestra es luchar porque la calidad del producto que aquí se haga sea de las mejores y la mejor posible."

Che

DECLARACIÓN DE AUDITORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los __ días del mes de _____ del año 2008.

_____	_____
Yadira Morales Alamo	Maikel L.Jimenez Moreira
Firma del autor	Firma del autor
_____	_____
Ing.Susel Cañete Pollán	MSc. David Leyva Leyva
Firma del tutor	Firma del tutor

_____

MSc Daymy Tamayo Ávila

Firma del tutor

MSc. David Leyva Leyva: Jefe de Departamento de Técnicas de Programación, Facultad 10, Universidad de las Ciencias Informáticas. Teléfono: (53) (07) 837 2511. davidl@uci.cu Licenciado en Cibernética-Matemática, Universidad de Las Villas (UCLV), 1988. Máster en Computación Aplicada, Universidad de Las Villas (UCLV), 1995.

Profesor de la Universidad de Holguín desde 1990. Jefe de Departamento de Informática (2003-2005). Participó en un Proyecto de Educación a Distancia en la Universidad del 2001 hasta el 2005, año en el que comenzó a trabajar en la Universidad de las Ciencias Informáticas. Ha participado en un gran número de eventos nacionales e internacionales. Le fueron concedidas sendas becas Intercampus en La Universidad de Castilla-La Mancha (1998) y La Universidad Autónoma de Madrid (2000). Trabajó como profesor invitado en la University of Belice (Belice, de 2001 a 2003) y de la Universidad Nacional de Ingeniería (Managua, Nicaragua, 2005).

Actualmente es líder del Polo Teleformación, grupo de investigación y desarrollo orientado a la extensión de la plataforma Moodle y al desarrollo y soporte de otras herramientas para la gestión de contenidos relacionados con el e-Learning.

MSc. Daymy Tamayo Ávila: Especialista de la Dirección de Teleformación, Profesora de Práctica Profesional V. Facultad 10, Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños Km. 2 ½, Torrens, Boyeros, Ciudad de La Habana. Cuba. Teléfono: (53) (07) 837 2453. daymy@uci.cu Ingeniera Informática, Universidad de Holguín, 2005. Máster en Informática Aplicada, UCI-CUJAE, 2007.

Culminó sus estudios obteniendo Título de Oro. Fue Alumna de Alto Aprovechamiento Académico y Alumna Ayudante en la Disciplina de Técnicas de Programación de Computadoras desde el segundo año de su carrera. Participó en un Proyecto de Educación a Distancia en la Universidad del 2001 hasta el 2005, año en el que comenzó a trabajar en la Universidad de las Ciencias Informáticas. Ha participado en un gran número de eventos nacionales e internacionales. Ha sido tutora de varios Trabajos de Curso y Diploma, y de Alumnos Ayudantes. Es la arquitecta principal del Polo Teleformación, grupo de investigación y desarrollo orientado a la extensión de la plataforma Moodle y al desarrollo y soporte de otras herramientas para la gestión de contenidos relacionados con el e-Learning.

Ing. Susel Cañete Pollán: Profesora del Departamento de Ingeniería y Gestión de Software en la facultad 10. scanete@uci.cu. Ingeniera en Ciencias Informáticas, Universidad de Ciencias Informáticas (UCI), 2007. Durante sus estudios fue alumna ayudante de las asignaturas Teleinformática, Metodología de Investigación y Práctica Profesional V, en la cual se desempeña actualmente. Participó en el proyecto Portales durante dos años.
En su primer año de trabajo ha sido tutora de varios trabajos de diploma y de una alumna ayudante. Trabaja en el Polo de Teleformación como miembro del equipo de calidad. Además es analista principal del proyecto "Sistema de Control de Documentos" del Ministerio de Energía y Petróleo de Venezuela.

AGRADECIMIENTOS

A **mis queridos padres** que de manera excepcional me han forjado, han sido mi ejemplo a seguir, mis fuerzas, mi ánimo, mi entereza, mi inspiración. Gracias por entenderme como nadie, por sus noches de desvelo, su dedicación constante, sus preocupaciones, por creer siempre en mí y principalmente por su imperecedero amor.

A mi abuelita Dora, por su consagrado apoyo, por ser mi mejor amiga, por sus oportunos consejos y por darme su mano cuando la necesité .Gracias por mimarme, entenderme y quererme tanto.

A mi hermano Yunior por ser el faro de mis pasos, la luz de mi vida, por ser incondicional y quererme con demasía. Gracias por ser tan especial. A mi familia en general por lo maravillosa que ha sido, por estar presente en todos los momentos de dificultad y de alegría de mi vida, en especial a mi tía María, mi hermano Yorgan, a Eve, Inés, Ernestina, tío Toño.

A mis amigos, en especial Linnet, Marelis, Irina, Arelis, Yaisy, Kirenia, Ana, Yaumara, Janys, Bertha, Yuni, Raúl, Juan José, Sandy, Erik, Yoandy, Derick,Reymi y a todos mis compañeros de grupo y del proyecto. A mi amigo Alioscha por su infinito cariño, su optimismo y por darme fuerzas para seguir adelante. A Osvaldo por su constante preocupación y ayuda. Gracias por ser mi eterno profesor y guía.

A mis amigas hermanas Irianna, Dayani, la Chacha y Yadira por su paciencia, su incondicional apoyo y sobre todo por comprenderme tanto. Gracias por su amistad. A Yanko por darme las primeras ideas de cómo comenzar la tesis, por su disposición, así como a su familia por estar siempre pendientes de mí.

A mi compañero de tesis y amigo Maikel por su dedicación, esfuerzo y comprensión. A mis tutores David y Susel por su ayuda y orientación en este trabajo. A Basulto por sus recomendaciones.

A mi papá UCI Roberto y mi madrastra Sioma, los cuales han sido para mí un estandarte de virtudes. Gracias por cuidarme como su hija y brindarme su cariño. A mi tío Merodio y mi hermano Osvaldo por ser la mejor familia que he tenido en la Universidad.

A la UCI por ser la casa donde me formé como ingeniera y a todos aquellos que contribuyeron a que este trabajo se pudiera realizar. ***¡Gracias!***

Yadira Morales Alamo

Es difícil en tan poco espacio agradecer a todas las personas e instituciones que han hecho por mí durante diecisiete años de estudios. Pero hoy se hacen realidad mis sueños. Agradecerle a mi mamá por ser mi guía, por sus consejos, por su sacrificio, por darme las fuerzas necesarias para levantarme de los tropiezos de la vida, en fin por hacer de mí el hombre que hoy soy. Agradecerle a mi papá por sus consejos y su confianza en mí, a mi hermano por su cariño y apoyo. A mi esposa por darme su amor, por sus consejos, por estar a mi lado en los momentos buenos y malos de mi carrera. Agradecerles a mis tías, mis primos, y toda mi familia por sus consejos y por su ejemplo. Agradecerles a mis amigos que compartieron estos cinco hermosos años de la carrera, en especial a Yoandy por ser mi mejor profesor de la carrera, Alioscha mi hermano inseparable que me ayudó en todo lo que necesité. Agradecerle al grupo 10506 por su muestra de unidad en todo momento. Agradecerles a los tutores David y Susel por su ayuda y guía en la investigación, al profesor Yanko por ayudarnos con la búsqueda de información y orientarnos por el camino correcto. Agradecerle de manera incondicional a mi compañera de tesis Yadira por su confianza y apoyo en el desarrollo de este trabajo. Agradecerle al compañero Fidel Castro y a la revolución por haber creado este proyecto.

A todos ***mil gracias***.

Maikel L.Jimenez Moreira

DEDICATORIA

A mis padres por la obra que han venido realizando, a los que me enseñaron a dar mis primeros pasos, a amar, a soñar, a proponerme metas, a lograr objetivos, a los que lograron hacer de mí una mujer de bien, a los que ni dedicándole mi vida recompensaré jamás. Por y para ello es este trabajo.

A mi abuelita Dora por ser el artífice principal de mis pasos, por trasmitirme su fuerza y darme su inmenso amor, porque hoy, aunque ausente, su recuerdo me acompaña, me guía, y me da fuerzas en los momentos difíciles.

A mi hermano Yunior por mostrarme siempre el camino, por estar a mi lado, por comprenderme y quererme tanto.

Yadira

A mi mamá por su dedicación, por sus consejos, por su cariño y sobre todo por creer en mí.

A mi hermano por ser la luz de mi vida y el inspirador de mis sueños.

A mi esposa por darme su amor y estar junto a mí brindándome su apoyo y confianza.

Maikel

RESUMEN

Este trabajo se enmarca dentro de la Ingeniería de Software, disciplina que tiene como objetivo proporcionar teorías, métodos y herramientas para el desarrollo de software de calidad.

El empleo de los métodos de desarrollo de software tradicionales ha causado un detrimento de la calidad de los productos software, en especial de los factores externos de calidad. Esto se debe en gran medida a la imposibilidad de probar el sistema hasta que se disponga de una implementación del mismo.

En esta investigación se ha realizado un estudio de la utilización de las técnicas basadas en las matemáticas como vehículo para crear especificaciones de sistemas con una sintaxis y semántica formalmente definida que permita 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, centrándose en las técnicas formales y herramientas que automatizan éstas.

Como resultado de este trabajo se presenta una propuesta de las técnicas formales y las herramientas basadas en ellas, que más se ajustan al proceso de desarrollo de software de la UCI, con el objetivo de aumentar la corrección, eficiencia e integridad del software.

PALABRAS CLAVE:

Corrección, eficiencia, integridad, técnicas formales, herramientas, desarrollo software.

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El desarrollo del software[1] a nivel mundial se ha visto afectado por el avance de las tecnologías que cada día se desarrollan más, apareciendo nuevos y exigentes sistemas informáticos. Esto requiere que los ingenieros que en esta área se especializan, deban estar al tanto de los sucesos que diariamente aparecen como mejor forma de desarrollo de la ciencia.

La industria del software ha estado estrechamente vinculada a todas las ramas de la economía y ha sido el suceso de trascendencia que ha contribuido al desarrollo de la era que vive el hombre de estos tiempos. Cuba no ha quedado fuera de estas aspiraciones y en todo momento se ha trazado la meta de informatizar la economía previendo los enormes resultados que esto trae consigo al desarrollo del país.

Aunque construir un sistema informático no es tan fácil como se cree, la experiencia demuestra que vertiginosamente aparecen programas más complejos y difíciles de construir. Aun así el equipo de desarrollo no puede perder como su objetivo primordial el crear un sistema que satisfaga las necesidades de los clientes o usuarios, haciéndose más trabajoso con las exigencias que persiguen estos productos[2] en la actualidad. La calidad como meta a lograr por cada equipo de trabajo, requiere de un proceso de desarrollo que garantice la corrección, eficiencia e integridad de cada aplicación que se produzca.

La presencia eficiente de estos elementos en el software, lo califican como de óptima calidad, aunque es un propósito que todavía dista de ser realidad para muchos desarrolladores en el mundo. Esto se debe en gran medida a la forma de trabajo en que se desempeñan los grupos de desarrollo para producir los sistemas informáticos. Las técnicas tradicionales han constituido la base para el desarrollo de sistemas que han arrastrado consigo errores hasta las últimas fases de su producción. Esto propició que se buscaran nuevas alternativas para garantizar más calidad en el software producido, es lo que se conoce en el ámbito informático como las técnicas formales. Métodos que han sido muy beneficiosos en los bancos, la industria del transporte aéreo y espacial. A pesar de sus ventajas potenciales en correspondencia con los métodos tradicionales no han sido muy utilizadas, en gran

[1] Es un término genérico que designa al conjunto de programas de distinto tipo (sistema operativo y aplicaciones diversas) que hacen posible operar con la computadora.
[2] Se refiere a productos software

medida, a la inexistencia de herramientas que permitan su aplicación práctica y a su desconocimiento por parte de los desarrolladores.[AVENDAÑO 2006]

El desarrollo de software de calidad en estos tiempos constituye un reto para los ingenieros. Por ello es que se ha convertido en una necesidad la existencia de los factores de calidad en el desarrollo de los mismos, en especial de la corrección, eficiencia e integridad por la importancia que su concepto encierra.

Producir software con calidad requiere de técnicas eficientes que garanticen la realización de las especificaciones planteadas por los involucrados y que en todo el proceso de desarrollo se puedan determinar a tiempo los errores que posea el software.

La producción de software en la Universidad de las Ciencias Informáticas (UCI) no está ajena a las necesidades antes planteadas. El proceso productivo está organizado en grupos de proyectos, que utilizan técnicas, métodos y metodologías para el desarrollo de software; algunas de ellas se han adaptado a las necesidades de los proyectos. Sin embargo, esto resulta insuficiente a pesar de que se han obtenido productos de calidad. Además, en algunas ocasiones no se dedica suficiente tiempo ni recursos a la captura de los requisitos, lo cual evitaría que se cometan errores en etapas futuras. Por otro lado, es propósito de la universidad producir software de calidad, para lo cual es de vital importancia la investigación sobre técnicas actuales y efectivas para la elaboración del software con calidad. En un trabajo anterior [CAÑETE 2007] se abordaron las técnicas formales, pero ésta sólo se enfocó al estudio del factor fiabilidad del software a lo largo de todo el ciclo de vida del software.

Por tanto se plantea el **problema científico** mediante la interrogante: ¿Cómo contribuir a mejorar la corrección, eficiencia e integridad del software que se desarrolla en la Universidad de las Ciencias Informáticas? Enmarcándose el **objeto de estudio** en la corrección, eficiencia e integridad del software y como **campo de acción** las técnicas formales para la corrección, la eficiencia e integridad del software.

Las **preguntas científicas** propuestas son:

- ¿Qué técnicas se emplean actualmente para garantizar los factores externos de calidad del software en la UCI?
- ¿Cómo se garantiza la corrección, eficiencia e integridad de un producto en los proyectos de la universidad?
- ¿Qué técnicas formales pueden ser aplicadas en los proyectos productivos de la UCI para garantizar la corrección, eficiencia e integridad?
- ¿Qué herramientas basadas en técnicas formales pueden ser utilizadas para garantizar los factores externos de calidad?

El **objetivo general** de la investigación es: Proponer técnicas formales y herramientas basadas en ellas, que contribuyan al aumento de la corrección, eficiencia e integridad en la producción de software de la Universidad de las Ciencias Informáticas.

Para esto se hace necesario abordar las siguientes **tareas de investigación**:

- Hacer un estudio sobre temas de calidad del software referente a factores externos.
- Analizar las técnicas formales aplicadas en el proceso de desarrollo de software que garanticen la corrección, la eficiencia e integridad del mismo.
- Valorar herramientas basadas en técnicas formales para determinar su posible utilidad en la obtención de software con calidad.
- Realizar una propuesta de técnicas formales y herramientas que contribuyan a garantizar la corrección, la eficiencia e integridad del software producido en los proyectos productivos de la UCI.

Para el cumplimiento de estas tareas se utilizaron los siguientes métodos teóricos:

El **Analítico-Sintético,** el cual se empleó para realizar un detallado análisis de temas de calidad del software, profundizando en los factores externos como la corrección, la eficiencia e integridad, así como de las técnicas formales y herramientas usadas para garantizar estos factores.

El **Inductivo-Deductivo** ya que se valoran temas de investigación de forma general para arribar a objetivos específicos dentro de la investigación. Por ejemplo, al tratar los temas de la calidad y sus factores, haciendo énfasis particularmente en la corrección, eficiencia e integridad.

El Análisis Histórico-Lógico para el estudio del desarrollo de la calidad del software, específicamente en factores externos como corrección, eficiencia e integridad, pues este método permite estudiar de forma analítica la trayectoria histórica real de los fenómenos, su evolución y desarrollo.

En cuanto a los métodos **empíricos** se hizo uso de la **encuesta,** con el propósito de recopilar información sobre el uso de las técnicas formales en la UCI y de la forma en que se emplean los métodos en la producción de software para contribuir a la calidad de los proyectos de la UCI; y la **entrevista** con el objetivo de conocer cómo se llevan a cabo las actividades de aseguramiento, construcción y mantenimiento del sofware.

El documento está estructurado en tres capítulos, además de las secciones de Glosario de Términos, Referencias Bibliográficas, Bibliografía Consultada y Anexos.

En el **Capítulo I** se expone la fundamentación teórica del tema. Incluye una descripción detallada del objeto de estudio, con la finalidad de comprender el proceso de calidad del software[3]. Se presenta el estado del arte de la calidad del software, además se hace un resumen de las técnicas formales y herramientas utilizadas en el proceso de desarrollo del software.

El **Capítulo II,** La corrección, la eficiencia e integridad del software, está centrado en los factores externos de calidad: corrección, eficiencia e integridad del software, vinculándolos al 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, basadas en estas técnicas, que contribuyen a garantizar la corrección, la eficiencia e integridad del software.

En el **Capítulo III** se analizan los resultados de la investigación teniendo como referencia las encuestas y entrevistas realizadas. Se ofrece una propuesta de técnicas formales y herramientas para ser aplicadas en los proyectos productivos de la UCI, las cuales garantizan la corrección, la eficiencia y la integridad durante todo el ciclo de vida del software.

[3] Algunos conceptos aparecerán textualmente porque es considerado necesario.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1 INTRODUCCIÓN

El presente capítulo tiene como objetivo exponer los fundamentos teóricos generales que sirven de punto de partida para garantizar los factores externos de calidad del software, en especial la corrección, eficiencia e integridad, siendo este el campo en el cual se desarrolla este trabajo. Se relacionará un conjunto de conceptos y elementos relacionados con el desarrollo del presente trabajo de diploma. Se hará referencia de forma general a varias técnicas tradicionales y formales que se utilizan actualmente en la construcción de software, así como herramientas basadas en métodos formales, cuya utilización es importante para obtener un producto con la calidad que se requiere.

1.2 PANORAMA HISTÓRICO

Desde los años 40 la calidad va ascendiendo considerablemente y su importancia es cada vez mayor. Lo desarrolladores en sus inicios se dedicaron a la detección de defectos tratando de controlar la calidad, pero a partir de los años 80 ya las investigaciones se centraron en el aseguramiento de la calidad, previniendo entonces los defectos y verificando el software a lo largo de su ciclo de vida. [ROMERO 2007]

Actualmente los esfuerzos se dedican a la gestión de la calidad para lograr la calidad total del producto a través de un mejoramiento continuo. Con todos estos cambios que se fueron desarrollando, que incluye a la industria del software, los clientes se ha vuelto cada vez más exigentes en cuanto a la calidad y complejidad de los productos informáticos, y lo que ha provocado que se haga cada vez más difícil cumplir con las expectativas de los usuarios finales por lo que se hace necesario buscar nuevas alternativas para tratar de resolver las dificultades existentes con el software.

Con el objetivo de dar solución a esta problemática se han venido desarrollando una serie de herramientas, técnicas y modelos que faciliten generar productos que cumplan con dichas expectativas e incluso que las rebasen, técnicas que prometen ser la solución a los problemas de calidad, costo y tiempos de desarrollo.

Poco a poco se fueron desarrollando nuevos métodos como fueron los métodos formales, que matemáticamente o mediante reglas lógicas, definían un sistema y podían comprobarlo disminuyendo

así los índices de errores que tenía. Pero a pesar de sus potenciales ventajas como es el aumento de la productividad muchos ingenieros no se atrevían a asumirlos por sus bases matemáticas, ya que para trabajar con ellos hay que tener conocimientos muy profundos de esa materia. [ARTUR BORONAT 2003]. Sin embargo otros decidieron esforzarse y darle la entrada a estos métodos ya que consideraron que era mucho menor el tiempo dedicado para el estudio y aprendizaje de los mismos que las perdidas por no emplearlos.

1.3 ¿QUÉ ES LA CALIDAD?

Hoy día con el vertiginoso avance de las tecnologías la calidad se ha convertido en un factor importante para la producción de software. Muchas personas se dedican a mejorar cada vez más la calidad y lo hacen de diferentes formas, una de ellas es definir un concepto que sirva de guía para llevar a la práctica tan importante conjunto de características. Debido a la alta importancia que tiene la calidad varios autores han formulado al respecto sus propios conceptos.

La Norma ISO 8402 define **calidad** como la totalidad de las características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades expresadas o implícitas, [8402 2002], por su parte, el Diccionario de la Real Academia Española, conceptúa la **calidad** como "propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor", y es sinónimo de "buena calidad" la "superioridad o excelencia". [Infocalidad 2004]

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

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”. [INDUSTRIAL and 1999]

La calidad no se ha convertido únicamente en uno de los requisitos esenciales del producto sino que en la actualidad es un factor estratégico clave del que dependen la mayor parte de las organizaciones, no sólo para mantener su posición en el mercado, también para asegurar su supervivencia.

1.3.1 Conceptos asociados

Para la comprensión del tema y del documento en general es importante conocer estos conceptos que a continuación se muestran [Infocalidad 2004]:

- **Control de calidad:** Técnicas y actividades de carácter operativo utilizadas para satisfacer los requisitos relativos a la calidad.
- **Aseguramiento de la calidad:** Conjunto de acciones planificadas y sistemáticas que son necesarias para proporcionar la confianza adecuada de que un producto o servicio satisfará los requisitos dados sobre la calidad.
- **Política de calidad:** Directrices y objetivos generales de una empresa relativos a la calidad, expresados formalmente por la dirección general.
- **Gestión de la calidad:** Aspecto de la función general de la gestión que determina y aplica la política de la calidad.
- **Sistema de calidad:** Conjunto de la estructura de organización de responsabilidades, de procedimientos, de procesos y de recursos que se establecen para llevar a cabo la gestión de la calidad.
- **Administración de la calidad.** Conjunto de actividades de la función general de administración que determina la política de calidad, los objetivos, las responsabilidades y la implementación de estos por medios tales como planeación de la calidad, el control de calidad, aseguramiento de la calidad y el mejoramiento de la calidad dentro del marco del sistema de calidad.
- **Planeación de la calidad.** Son las actividades que determinan los objetivos y requisitos para la calidad, así como los requisitos para la implementación de los elementos del sistema de calidad.
- La filosofía de la **Calidad Total** proporciona una concepción global que fomenta la Mejora Continua en la organización y la involucración de todos sus miembros, centrándose en la satisfacción tanto del cliente interno como del externo. Se puede definir esta filosofía del siguiente modo: Gestión (el cuerpo directivo está totalmente comprometido) de la Calidad (los requerimientos del cliente son comprendidos y asumidos exactamente) Total (todo miembro de la organización está involucrado, incluso el cliente y el proveedor, cuando esto sea posible). [GONZALEZ 2007]

1.4 ¿QUÉ ES LA CALIDAD DEL SOFTWARE?

Uno de los problemas que se afrontan actualmente en la esfera de la informática es la calidad del software. Desde la década del 70 este tema ha sido motivo de preocupación para ingenieros e investigadores de software, los cuales han realizado gran cantidad de investigaciones respecto a cómo obtener un producto con calidad y cómo garantizar la calidad del software.

La calidad del software es definida por [S.PRESSMAN 1998] como la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se esperan de todo software desarrollado profesionalmente.

De acuerdo a la definición del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, Std. 610-1990) “La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario”.

Según [UNIVERSIDAD 1990] es la concordancia del software producido con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente.

La calidad del software es medible y varía de un sistema a otro o de un programa a otro. Un software elaborado para el control de naves espaciales debe ser confiable al nivel de "cero fallas", al igual que el software para un banco el nivel de seguridad debe ser máximo ; sin embargo un software hecho para ejecutarse una sola vez no requiere el mismo nivel de calidad; mientras que un producto de software para ser explotado durante un largo período (10 años o más), necesita ser confiable, robusto y flexible para disminuir los costos de mantenimiento y perfeccionamiento durante el tiempo de explotación.

La calidad del software según [CARRASCO 1995] puede medirse después de elaborado el producto. Pero esto puede resultar muy costoso si se detectan problemas derivados de imperfecciones en el diseño, por lo que es imprescindible tener en cuenta tanto la obtención de la calidad como su control durante todas las etapas del ciclo de vida del software.

[QUESADA. 2001] plantea que es el conjunto de cualidades que la caracterizan y que determinan su utilidad y existencia. La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad.

Conforme al marco conceptual para el modelo de calidad especificado por la norma ISO/IEC 9162, la calidad de software comprende tres enfoques íntimamente relacionados: proceso, producto y efecto del software. [QUESADA. 2001]

De lo anteriormente dicho se puede inferir que un producto software con calidad debe tener relacionados enfoques de proceso, producto y muy importante el efecto que este tenga en el cliente, además debe garantizar que sea aceptado por usuarios satisfechos ya que respondan a sus necesidades, no presenten errores, sean confiables y de fácil entendimiento.

1.4.1 Tipos de calidad de software

- **Calidad de diseño**

La calidad de diseño [S.PRESSMAN 1998] se refiere a las características que pueden especificarse para cada diseño. El grado de materiales, tolerancia y especificaciones contribuyen a la calidad de diseño. Comprende las especificaciones y el diseño del sistema. Si se utilizan materiales de altos grados y se especifican tolerancias más estrictas y niveles más altos de rendimiento, la calidad de rendimiento aumenta, si el producto se fabrica de acuerdo con esas especificaciones.

- **Calidad de concordancia**

La calidad de concordancia [S.PRESSMAN 1998] es el grado de cumplimiento de las especificaciones de diseño durante su realización. Es un aspecto centrado en la implementación. Cuanto mayor sea el grado de cumplimiento, mayor será el nivel de calidad de concordancia.

1.4.2 Control de la calidad del software

El control de la calidad del software o en inglés, *Software Quality Control* son las técnicas y actividades de carácter operacional, utilizadas para satisfacer los requisitos relativos a la calidad, centradas en dos objetivos fundamentales:

- Mantener bajo control un proceso.
- Eliminar las causas de los defectos en las diferentes fases del ciclo de vida.

Las técnicas de Ingeniería de Software para conseguir calidad en el software se denominan Garantía de Calidad del Software[4], o en inglés, *Software Quality Assurance*[5]. [MESTRAS 2004]

La Garantía de Calidad del Software engloba:

- Enfoque de gestión de calidad.
- Tecnologías de Ingeniería de Software (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 Ingeniería de Software.
- Mecanismos de medición y generación de informes.

De forma general se puede decir que son las actividades para garantizar la calidad de los productos desarrollados.

1.4.3 Estándares y Modelos Internacionales para la calidad del software

IEEE (Instituto de Ingenieros Eléctricos y Electrónicos)

Cuando se habla de estándares para la calidad del software no se puede dejar de mencionar el estándar IEEE 730-1998 para el Plan de Aseguramiento de la Calidad de Software [IEEE 1998].Este aborda aspectos como la administración, la documentación, el control de código, entre otros. Dentro de la documentación se le adjudica gran importancia al Plan para la Gestión de Configuración de Software

[4] Garantía de Calidad del Software(GCS)

reflejado en el estándar IEEE 828-1998 [IEEE 1998]. Dicho plan trata todo lo referente a la asignación de las responsabilidades, la identificación de las actividades que se realizarán durante todo el proceso, la identificación de la configuración, el reconocimiento de los elementos de configuración, el control de la configuración, el acceso a las bibliotecas, la aprobación o desaprobación de un cambio y la implementación del cambio de ser aprobado (todas éstas incluidas entre las tareas de la Gestión de la Configuración).

CMMI (Modelo de Capacidad y Madurez Integrado)

CMMI (Modelo de Capacidad y Madurez Integrado) es un modelo que constituye un marco de referencia de la capacidad de desarrollo de software de diferentes empresas, siendo una base para evaluar la madurez de las mismas y una guía para llevar a cabo una estrategia de mejora continua. Este modelo surgió a partir de CMM (Modelo de Capacidad y Madurez) como resultado de la integración de varios modelos que definían la madurez en diferentes disciplinas, pero que dada la variedad constituían un problema para las empresas por lo que fue necesario agruparlos en el CMMI.

La capacidad del proceso de software describe el rango de resultados esperados que se obtienen siguiendo un proceso de software, mientras que el desempeño representa los resultados reales obtenidos. La madurez del proceso de software se refiere a cuán explícitamente definido, administrado, medido, controlado y efectivo ha sido un proceso en específico.

CMMI dirige su enfoque a la mejora de procesos en una organización, los estudia y mide la capacidad para construir un software de calidad, atendiendo a una escala de cinco niveles (inicial, repetible, definido, dirigido y optimizado). Para que cada organización pueda enfocar la mejora de sus procesos se especifica en cada nivel de madurez un conjunto de áreas de proceso, que se describen en términos de prácticas, estas son un conjunto de actividades que contribuyen a la implementación eficiente de un área de proceso, si se cumplen todas las prácticas y se satisfacen todas las áreas de proceso de un determinado nivel entonces la empresa podrá certificarse en ese nivel de madurez. [CMMIv1.2 2006]

En CMMI como en otros modelos de calidad se tiene en cuenta las mediciones como datos necesarios para el cálculo de métricas. Estas deben ser recepcionadas por cada desarrollador de software en cualquier proyecto, o en general por el equipo de trabajo.

[5] Software Quality Assurance (SQA)

ISO (Organización Internacional de Normalización)

ISO es una red de institutos nacionales de estándares de 156 países que promueve la normalización internacional para facilitar el intercambio de bienes y servicios como de aplicaciones. Relacionado con los procesos de software ha sido implementado el modelo de evaluación y mejora del proceso de software ISO 9000, dicho nombre genérico es con el que coloquialmente se designa a una familia de normas de aseguramiento de la calidad.

La serie ISO 9000 esta formada por cinco documentos, tres de ellos son modelos de aseguramiento de la calidad, el específicamente el 9001, 9002 y 9003. Los otros dos son simples lineamientos que sirven de apoyo [ISO/IEC 2005].

La serie ISO, es una de las más reconocidas e importantes y provee normas para servicios y productos, en todos los esferas del quehacer humano (Ingeniería, Biología, Medicina, entre otras). [INDUSTRIAL and 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 [INDUSTRIAL and 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. [INDUSTRIAL and 1999]

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. Según [INSTITUTO 2006] la ISO 9001:2000 es la versión más actual y ha sido adoptada como modelo a seguir para obtener la certificación de calidad.

ISO 9002 Modelo de aseguramiento de la calidad aplicable a la fabricación y a la instalación.

ISO 9003 Modelo de aseguramiento de la calidad aplicable a la inspección y a ensayos finales.

ISO 9004 Principios, conceptos, lineamientos para la gestión de calidad y elementos del sistema de calidad.

1.4.4 Aseguramiento de la calidad del software

El aseguramiento de la calidad del software es el conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza en que el producto (software) satisfará los requisitos dados de calidad. El aseguramiento de calidad del software se diseña para cada aplicación antes de que se comience a desarrollar y no después. [CUEVA 1999]

Algunos autores prefieren decir garantía de calidad en vez de aseguramiento, pero el término garantía puede traer confusiones con la garantía del producto, sin embargo el aseguramiento pretende dar seguridad y confianza de que el producto tiene la calidad requerida.

El aseguramiento de calidad del software está presente en:

- Métodos y herramientas de análisis, diseño, programación y prueba.
- Inspecciones técnicas formales en todos los pasos del proceso de desarrollo del software.
- Estrategias de prueba multiescala.
- Control de la documentación del software y de los cambios realizados.
- Procedimientos para ajustarse a los estándares y dejar claro cuando se está fuera de ellos.
- Mecanismos de medida (métricas).
- Registro de auditorías y realización de informes.

Actividades para el aseguramiento de calidad del software

- Métricas de software para el control del proyecto.
- Verificación y validación del software a lo largo del ciclo de vida.
    - Incluye las pruebas y los procesos de revisión e inspección.
- La gestión de la configuración del software.[CUEVA 1999]

1.4.5 ¿Cómo obtener un software de calidad?

Antes de comenzar a explicar cómo obtener un software de calidad es importante conocer características que debe cumplir dicho software.

Un software de calidad debe [I-SOL 1999] **:**

- Ser eficaz, es decir, debe ser capaz de realizar las funciones establecidas, de ser amigable. Un usuario debe utilizar el software porque produce resultados confiables. Realiza todas las operaciones que se requieren, ejecuta las operaciones en un tiempo aceptado y es fácilmente usado por el grupo de usuarios a quien esté dirigido.
- Ser eficiente, es decir, 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 uso obtengan el máximo beneficio o por lo menos un beneficio aceptable en un período de tiempo establecido.

La obtención de un software con calidad implica la utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software.

La política establecida debe estar sustentada sobre tres principios básicos: tecnológico, administrativo y ergonómico. El principio tecnológico define las técnicas a utilizar en el proceso de desarrollo del software.

El principio administrativo contempla las funciones de planificación y control del desarrollo del software, así como la organización del ambiente o centro de ingeniería de software.

El principio ergonómico define la interfaz entre el usuario y el ambiente automatizado.

La adopción de una buena política contribuye en gran medida a lograr la calidad del software, pero no la asegura. Para el aseguramiento de la calidad es necesario su control o evaluación.

1.4.6 ¿Cómo controlar la calidad del software?

Para lograr un control eficiente de la calidad del software es necesario, definir los parámetros, indicadores o criterios de medición, ya que como bien plantea Tom De Marco, " usted no puede controlar lo que no se puede medir".[FERNÁNDEZ CARRASCO 1995]

Muchos autores han definido cualidades para medir la calidad del software, los cuales las denominan y clasifican de diversas formas. Por ejemplo, John Wiley define métricas de calidad y criterios, donde cada métrica se obtiene a partir de combinaciones de los diferentes criterios. La Metodología para la evaluación de la calidad de los medios de programa en Rusia, define indicadores de calidad estructurados en cuatro niveles jerárquicos: factor, criterio, métrica y elemento de evaluación, donde cada nivel inferior contiene los indicadores que conforman el nivel precedente.

Otros autores identifican la calidad con el nivel de complejidad del software y definen dos categorías de métricas: de complejidad de programa o código, y de complejidad de sistema o estructura. Un factor importante radica en que todos los autores coinciden en que el software posee determinados índices medibles que son las bases para la calidad, el control y el perfeccionamiento de la productividad. Una vez que se seleccionan los índices de calidad se debe establecer el proceso de control, el cual requiere los siguientes pasos: [FERNÁNDEZ CARRASCO 1995]

- Definir el software que va a ser controlado: clasificación por tipo, esfera de aplicación, complejidad, entre otros, de acuerdo con los estándares establecidos para el desarrollo del software.
- Seleccionar una medida que pueda ser aplicada al objeto de control. Para cada clase de software es necesario definir los indicadores y sus magnitudes.

- Crear o determinar los métodos de valoración de los indicadores: métodos manuales como cuestionarios o encuestas estándares para la medición de criterios periciales y herramientas automatizadas para medir los criterios de cálculo.
- Definir las regulaciones organizativas para realizar el control: quiénes participan en el control de la calidad, cuándo se realiza, qué documentos deben ser revisados y elaborados.

Todo lo antes planteado tiene un único fin y es lograr que el producto que se elabore sea de calidad.

1.5 FACTORES DE CALIDAD DEL SOFTWARE

Al referirse a la calidad del software se debe tener presente que la misma se divide en calidad del producto y en calidad del proceso de desarrollo del producto. 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 software.

La calidad de un producto tiene muchos factores en su producción para ofrecer al consumidor lo que realmente necesita del producto para satisfacer sus necesidades. Los factores de calidad son características que sirven para medir un software, 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. Estos atributos se dividen en internos y externos. Los atributos internos pueden ser medidos directamente en términos del producto, proceso o recurso. Por su parte los atributos externos no pueden ser medidos directamente sino teniendo en cuenta cómo la entidad se relaciona con el ambiente en que se desarrolla.

Los atributos internos por lo general son indicadores de los atributos externos. Un atributo externo puede estar influenciado por más de un atributo interno y a su vez un atributo interno puede influir en uno o más atributos externos.

1.5.1 Factores Internos

Son cualidades aplicables a un producto de software perceptibles mayormente por los programadores ya que tienen acceso al código fuente. Por ejemplo la modularidad y la legibilidad.

1.5.2 Factores Externos

Son cualidades cuya presencia y ausencia en un producto de software puede ser detectada por los usuarios del software. Estos factores son el centro de atención de los desarrolladores, sin embargo la clave para comprenderlos radica en los atributos internos.

Los factores externos son clasificados según diferentes autores de diversas formas. McCall y sus colegas [J. MCCALL 1977], los cuales son autores reconocidos y su bibliografía es confiable ya que se puede encontrar en el [S.PRESSMAN 1998] los subdividen en tres importantes grupos (ver anexo 1 y 2):

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 [CUEVA 1999]

El factor **fiabilidad** es el grado con el que se espera que una aplicación lleve a cabo las operaciones especificadas y con la precisión requerida. 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 éste." [CABRERA 2006] Para darle claridad a este concepto hay que tener en cuenta que un fallo es cualquier falta 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. [S.PRESSMAN 1998].Este factor fue estudiado en el trabajo anterior [CAÑETE 2007], donde se analizaron 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 y 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.

La **usabilidad** expresa la facilidad con que las personas con diferentes formaciones tanto culturales como pedagógicas y distintos comportamientos pueden aprender a utilizar los productos de forma cómoda, interactuar con ellos 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 detalladas que sirvan para los usuarios novatos y no afecten al resto de las personas que están más experimentadas sobre este tema con largas y específicas explicaciones.

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.

Una de las claves segura para la facilidad de uso es la simplicidad estructural. Un sistema bien diseñado, construido siguiendo una estructura clara y correctamente pensada, tiende a ser más fácil de aprender y usar que uno confuso.

Los buenos diseñadores de interfaces siguen una política prudente, hacen las menos suposiciones posibles sobre los usuarios, sin embargo se espera que los mismos sepan las operaciones básicas e imprescindibles como es la lectura, el trabajo con el mouse, presionar un botón y teclear (lentamente); no mucho más. Si el software está dirigido a un área especializada de aplicación, se puede dar por supuesto que los usuarios están familiarizados con sus conceptos básicos, pero incluso esto es arriesgado.

Otro factor importante dentro de la calidad del software es la **facilidad de mantenimiento** [MEDINA PATÓN 2007], este se realiza cuando ya ha sido entregado el software. El mantenimiento del software es definido como el conjunto de medidas que hay que tomar para que el sistema siga trabajando correctamente. Este proceso cubre todo el ciclo de 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.

Entre las características sobresalientes del mantenimiento del software se destacan:

- El software no envejece.
- El mantenimiento del software supone adaptar el paquete o sistema objeto del mismo a nuevas situaciones como cambio de hardware o cambio de software (S.O).
- Todo sistema software conlleva mejoras o añadidos indefinidamente.

Al cerrar todo proyecto se debe considerar y prever las normas del mantenimiento del sistema (tanto en connotaciones hardware como software).

Se estima que el 70% del costo del software se dedica al mantenimiento. El coste del mantenimiento de un producto software a lo largo de su vida útil es superior al doble de los costes de su desarrollo. Lientz identifica ocho categorías en que se puede desglosar el mantenimiento:

- Cambios en los requisitos del usuario (41.8%)
- Cambios en los formatos de los datos (17.6%)
- Cambios de emergencia (12.4%)

- Arreglo de rutinas (9%)
- Cambios en el hardware (6.2%)
- Documentación (5.5%)
- Mejoras en la eficiencia (4%)
- Otros (3.5%)

Existen 4 tipos de mantenimiento:

- **Correctivo:** Son las actividades que corrigen defectos y fallos detectados tanto en el hardware como en el software, son detectados mientras el sistema está en explotación por el usuario.
- **Adaptativo:** Consiste en la modificación de un programa debido a cambios en el entorno (hardware o software) en el que se ejecuta.
- **Perfectivo:** Actividades que se realizan para mejorar o añadir nuevas funcionalidades al sistema requeridas por el usuario.
- **Preventivo:** Modificación del software para mejorar las propiedades de dicho software (calidad y mantenibilidad) sin alterar sus especificaciones funcionales.

La **flexibilidad** se resume en esta pregunta: ¿Puedo añadir nuevas opciones? Se define como el esfuerzo requerido para modificar una aplicación en funcionamiento.

La **facilidad de prueba** como factor es el esfuerzo requerido para probar una aplicación de forma que cumpla con lo especificado en los requerimientos, o sea que se puedan probar todas las opciones del producto software que se está analizando.

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 ésta 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; un error bajo esta circunstancia sería fatal para el ingeniero que esté a cargo de dicha actividad.

En resumen la portabilidad es la facilidad con la que un software puede ser transportado sobre diferentes sistemas físicos o lógicos. [PAREDES 2005]

A la hora de definir la calidad de un producto software se debe tener presente si se puede utilizar alguna parte del mismo en otra aplicación, o sea que se pueda reutilizar. La **reutilización** es el grado en que partes de una aplicación pueden utilizarse en otras aplicaciones. 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 [FRANCISCO J. 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 eficiencia como en los demás factores de 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.

La **eficiencia** es uno de los factores más importantes en cuanto a calidad ya que un producto puede ser muy fácil de usar, muy reutilizable e incluso fiable pero si no es eficiente entonces ya no tiene la calidad que se espera o se desea, ya que la eficiencia es un factor que debe tratarse desde el comienzo del ciclo de vida del desarrollo del software hasta la etapa de mantenimiento del mismo, así como la **corrección** ya que en cada fase puede ocurrir uno o varios errores y no se podría cumplir con exactitud los requisitos del sistema. La **integridad** es otro factor significativo en la producción de software desde sus inicios. Estos tres factores van a ser tratados más adelante en el capítulo 2.

1.6 PROCESO DE DESARROLLO DEL SOFTWARE

El desarrollo de la industria de software es un proceso que implica que los productos realizados deban de ser confiables, precisos, rápidos de utilizar, flexibles y además deban de estar bien documentados. Un producto cumple con estos requisitos cuando se le han aplicado las técnicas de Ingeniería de Software adecuadas, se han utilizado los roles apropiados para el desarrollo de las tareas en la

empresa de software y en su etapa de comprobación se le han aplicado las pruebas necesarias para lograr el nivel de calidad requerido**.** [NAPAL. 2003]

1.6.1 Modelos del Proceso de Desarrollo Software

No existe consenso sobre cuál es el mejor modelo del proceso software. Distintos equipos de desarrollo pueden utilizar diferentes modelos de proceso software para producir el mismo tipo de sistema software. Sin embargo, algunos modelos son más apropiados para producir ciertos tipos de sistemas, de forma que si no se utiliza un modelo adecuado puede ocurrir que el sistema software resultante sea de menor calidad. [DUQUE 2000]

El reparto de costes entre las distintas fases del proceso de desarrollo es difícil de determinar dado los distintos modelos de proceso existentes. Sin embargo, en dependencia del modelo que se adopte, al menos el 60% del coste total se emplea en la actividad de evolución del sistema. La estimación de este porcentaje es pesimista, ya que la tasa de crecimiento de nuevos productos software es mucho mayor que la tasa de productos software que quedan en desuso (no tienen que ser mantenidos), por lo que el número de operaciones de mantenimiento que se realizan sigue aumentando. El proceso de diseño software debería, por tanto, tener en cuenta la posterior evolución del sistema.

Las características deseables de un proceso de desarrollo software son:

Claridad: El proceso de desarrollo es claro cuando se entiende con facilidad.

Visibilidad: Un proceso de desarrollo es visible cuando sus actividades producen resultados claros identificables externamente.

Facilidad de soporte: Exige disponer de herramientas CASE (*Computer-Aided Software Engineering*) que den soporte a todas o alguna de las actividades del proceso de desarrollo.

Fiabilidad: Un proceso de desarrollo es fiable cuando es capaz de detectar posibles errores.

Facilidad de mantenimiento: Requiere capacidad para incorporar nuevos requisitos o modificar alguno o algunos de los existentes.

Rapidez: Un proceso software es rápido cuando se puede obtener, a partir de la especificación, una implementación del sistema en un tiempo reducido.

Modelo en cascada o convencional: Tomado de otras ingenierías según [DUQUE 2000], es el primer modelo de desarrollo software propuesto. Ampliamente usado en la industria por su facilidad de gestión y visibilidad. En la figura 1 se representa el secuenciamiento de las actividades de este modelo de desarrollo.

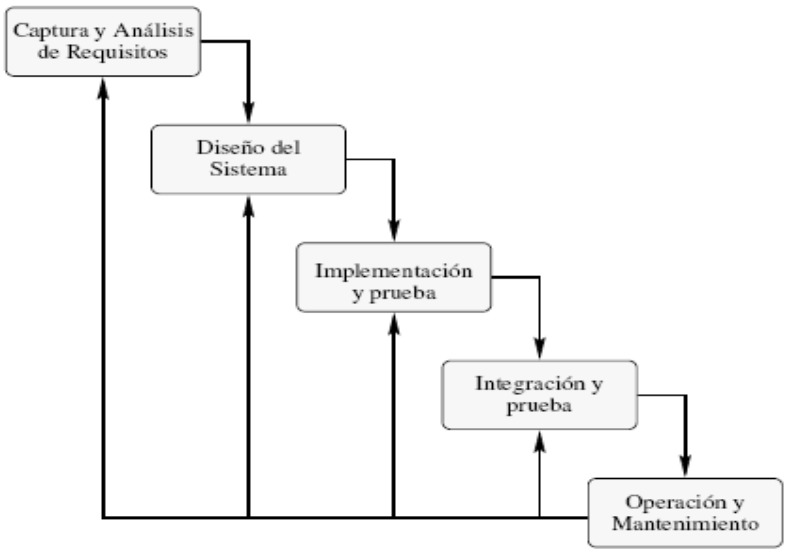


Figura 1: Modelo en cascada o convencional.

Sin embargo, su principal problema reside en su poca flexibilidad al separar el proceso de desarrollo en etapas totalmente distintas. En la práctica estas etapas no tienen fronteras tan bien definidas, lo que hace que, en no pocas ocasiones, se solapen y compartan información.

Los principales problemas de este modelo son: dificultad para realizar prototipos, reutilizar software y realizar pruebas sin disponer de una implementación del sistema.

Modelo evolutivo: En este modelo se entrelazan las actividades de especificación, desarrollo y validación. Inicialmente, se desarrolla rápidamente un sistema inicial a partir de una especificación muy abstracta. El sistema se va refinando con la información que van suministrando los clientes y/o usuarios hasta que se obtiene un sistema final que satisfaga todas las necesidades previstas. El sistema final obtenido puede rediseñarse para producir otro más robusto y más fácil de mantener. En la figura 2 se esquematiza este modelo.[GIL 1999]

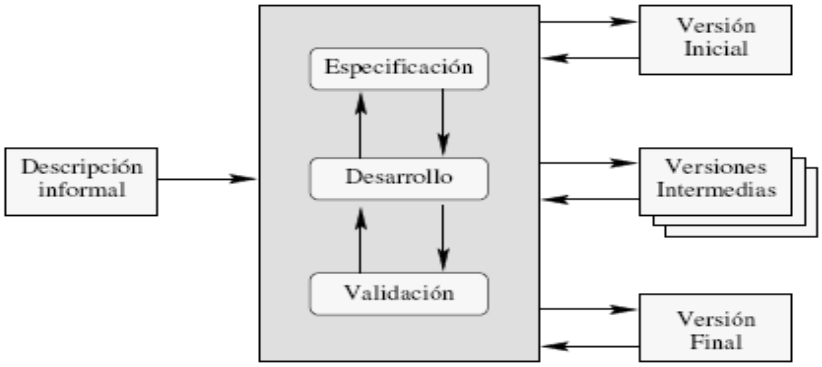


Figura 2: Modelo evolutivo.

Existen dos tipos de procesos de desarrollo evolutivos:

Exploratorio: Su objetivo es trabajar con el cliente para identificar y construir el sistema final a partir de una especificación informal. El resultado del proceso es el sistema final.

Prototipado desechable: Su objetivo es entender los requisitos del cliente. El resultado del proceso es la especificación del sistema (el prototipo se deshecha).

Los principales problemas de este modelo son: escasa visibilidad; los continuos cambios que hacen que los sistemas desarrollados estén deficientemente estructurados; y la necesidad de disponer, en muchos casos, de un equipo de desarrollo altamente calificado. Estos problemas hacen que la aplicación de este modelo se suela limitar a sistemas interactivos de tamaño pequeño o mediano. La deficiente estructura dificulta las tareas de mantenimiento de ahí que se suela aplicar a sistemas con una vida corta y a partes de grandes sistemas, especialmente a sistemas de inteligencia artificial y a interfaces de usuario.

Modelo transformacional: [DUQUE 2000] Se basa en disponer de una especificación formal del sistema y en transformar, con métodos matemáticos, esta especificación en una implementación. Si las transformaciones que se aplican son correctas es posible asegurar que el sistema construido satisface la especificación, es decir, es posible obtener programas correctos por construcción.

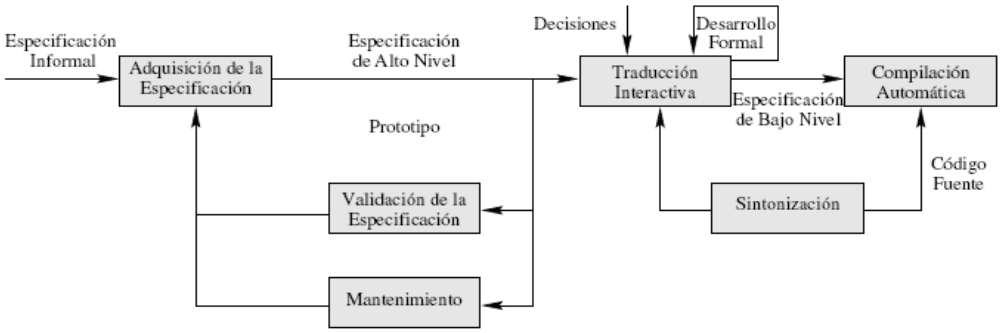


Figura 3: Modelo transformacional.

Otra de sus ventajas es la posibilidad de realizar el mantenimiento a nivel de especificación. Por lo que es necesario disponer de una especificación inicial correcta y de diseñadores altamente calificados. Además no existe apenas experiencia en la aplicación de este modelo a grandes proyectos.

Modelo basado en reutilización: En este modelo se supone que alguno de los componentes del sistema final ya existe. El proceso de desarrollo se centra en integrar las partes ya existentes más que en construir todo el sistema desde el principio.

Las ventajas que desde un punto de vista económico puede producir este modelo actualmente empiezan a ser estudiadas en profundidad. Prácticamente no existe experiencia sobre el empleo de este modelo, si bien, se están haciendo numerosos estudios e investigaciones para posibilitar su uso. [DUQUE 2000]

Modelo en espiral: [DUQUE 2000] Desarrollado por Boehm en el año 1988 con el objetivo de reunir las ventajas de los modelos de proceso software en cascada y de prototipado. Se incluye el análisis de riesgo como una parte importante del proceso de desarrollo software.

El modelo tiene la forma de una espiral (ver anexo 3) en la que cada vuelta representa cada una de las fases en las que se estructura el proceso software y está organizada en cuatro sectores:

1. Definición de objetivos, alternativas y restricciones de cada fase del proyecto.
2. Evaluación de alternativas y análisis de riesgos.
3. Desarrollo y validación. Se elige el modelo de proceso de desarrollo que se considere más adecuado.
4. Planificación de las siguientes fases del proyecto.

1.6.2 Técnicas utilizadas en el proceso de desarrollo del software

Los modelos tradicionales del ciclo de vida de un sistema software (modelo de ciclo de vida en cascada) muestran numerosas deficiencias cuando se aplican a sistemas complejos. Estas deficiencias son motivadas, en gran medida, por la imposibilidad de realizar pruebas hasta la fase de implementación. Ésta problemática se acentúa en los sistemas distribuidos, ya que estos son intrínsecamente más complejos que los sistemas secuenciales clásicos.

Las **técnicas tradicionales** como su nombre lo indica son las técnicas más usadas, las más habituales y antiguas en el proceso de desarrollo de un producto software. Éstas técnicas son importantes en el momento de obtener los requerimientos, ya que con el uso de ellas se extraen requisitos y un ejemplo de ello son las encuestas, cuestionarios y análisis de documentación. También se pueden extraer requerimientos mediante las entrevistas a personas individuales. Ellas pueden ser empleadas por personal no especializado o sin entrenamiento previo ya que son muy fáciles de aplicar. El esfuerzo necesario para aplicar una técnica tradicional puede ser cuantificado en términos de:

- Disponibilidad de los *stakeholders*[6]: cantidad de personas y el tiempo de disponibilidad.
- Trabajo de preparación de las sesiones.
- Entrenamiento de los ingenieros de requerimiento [COUTIN 2005]

Por otro lado se encuentran los **métodos formales o técnicas formales** las cuales permiten a los ingenieros de software especificar, desarrollar y verificar un sistema basado en computadora aplicando una notación rigurosa. Además permiten crear una especificación sin ambigüedades que sea más completa y constante que la que se utilizan en los otros métodos.

Según se plantea en [S.PRESSMAN 1998] 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 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 cliente comienza a utilizarlo.

[6] Personas u organizaciones que están activamente implicadas en el negocio ya sea porque participan en el o porque sus intereses se ven afectados con los resultados del proyecto .Pueden ser los propietarios, la dirección, los clientes, los trabajadores, los proveedores, etc.

Las técnicas formales también tienen sus propias desventajas, por ejemplo en el momento de hacer uso de ellas, 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 ARIAS 2000]

Usando estos métodos es más fácil descubrir y corregir los errores del software, principalmente en las etapas de especificación de requisitos y la de diseño ya que estas técnicas sirven como base para la verificación de programas y por consiguiente se detectan fallos que no son posibles ver de otra manera.

1.6.3 Herramientas empleadas para el desarrollo del software

Hoy día la informática ha adquirido un alto auge e importancia en las diferentes esferas de la sociedad, especialmente en la economía. También con el estudio de esta materia se ha elevado el nivel cultural y profesional de los ingenieros en este campo. Actualmente muchos de los informáticos que se dedican a la construcción de software utilizan herramientas de desarrollo basadas en tecnologías orientadas a objetos, las cuales contribuyen a una mejor reutilización del software.[CALIDAD 2008]

Las herramientas forman una parte fundamental del entorno de gestión de la calidad, ya que facilitan y ayudan en las diferentes etapas de dicho proceso. El uso de herramientas simplifican algunas tareas que anteriormente se les dedicaba mucho tiempo en su realización; y hacen posible la ejecución de otras, que serían muy difíciles de realizar de forma manual.

Existen una gran cantidad de herramientas en el mercado mundial, tanto comerciales como *Open Source*, que cubren todas las áreas en las que se puede gestionar la calidad del software. A continuación se representa un listado de algunas clases de herramientas existentes y su funcionalidad:

- **Herramientas de gestión de pruebas:** Dan soporte a varias actividades y tareas incluidas en las actividades de aseguramiento de la calidad a lo largo del ciclo de vida del desarrollo.
- **Herramientas de ejecución de pruebas:** Herramientas que permiten ejecutar pruebas. Entre ellas se incluyen las herramientas de automatización, las de pruebas de rendimiento y las de pruebas de seguridad, que permiten detectar errores de seguridad en etapas tempranas del ciclo de vida, como pueden ser intentos no autorizados para operar el software.

- **Herramientas de gestión de requisitos:** Herramientas que se encargan para grabar, gestionar y priorizar los requisitos de un sistema.
- **Herramientas para la gestión de defectos:** Herramientas que permiten realizar tareas de registro y seguimiento de errores a través del ciclo de vida.
- **Herramientas de gestión de la configuración:** Herramientas que se encargan del control de la evolución de los productos software.
- **Herramientas para pruebas unitarias:** Herramientas que se utilizan para probar unidades de código, una vez que están completas, para asegurar el correcto funcionamiento de las mismas.
- **Herramientas de inspección de código:** Herramientas que permiten verificar el estilo, seguridad y vulnerabilidades del código de una aplicación.
- **Herramientas de modelado/diseño:** Herramientas que permiten capturar, guardar, rechazar e integrar información automáticamente y hacer el diseño de la información.
- **Entornos integrados de desarrollo[7]:** Entornos de programación que han sido empaquetados como un programa de aplicación.

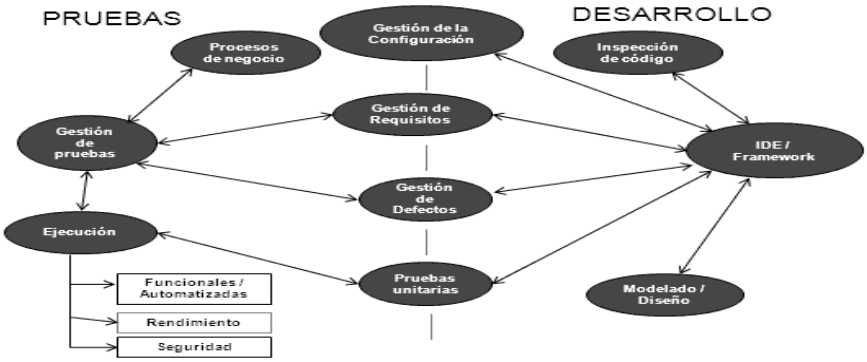


Figura 4: Mapa de herramientas.

En el gráfico de la figura 4 [CALIDAD 2008] se muestra una clasificación de herramientas en base a tres grupos: las relacionadas con las pruebas, las relacionadas con el desarrollo y las que están relacionadas con ambos dominios.

Como herramientas relacionadas con ambos dominios están las herramientas de gestión de la configuración, las de gestión de requisitos, las de gestión de defectos y las de pruebas unitarias.

[7] Entornos integrados de desarrollo (IDEs)

Con las pruebas están relacionadas las herramientas de procesos de negocio, las herramientas de ejecución de pruebas y las herramientas de gestión de pruebas. Las herramientas de gestión de pruebas abarcan varias tareas y actividades relacionadas con el aseguramiento de la calidad, como pueden ser las que realizan las herramientas con las que se les ha relacionado: herramientas de proceso de negocio, gestión de requisitos, gestión de defectos, pruebas unitarias y ejecución de pruebas.

Entre las herramientas orientadas al desarrollo se encuentran las de inspección de código, las de modelado (diseño) y los *IDE/Framework*. Estos últimos aparecen relacionados con las herramientas de inspección de código, las de modelado (diseño), las de gestión de la configuración, las de gestión de requisitos, las de gestión de defectos y las de pruebas unitarias.

Muchas empresas se han extendido a la adquisición de herramientas **CASE** (Ingeniería Asistida por Computadora), con el fin de automatizar los aspectos clave de todo el proceso de desarrollo de un sistema.

CASE proporciona un conjunto de herramientas semiautomatizadas y automatizadas que están desarrollando una cultura de ingeniería nueva para muchas empresas. Uno de los objetivos más importante del CASE (a largo plazo) es conseguir la generación automática de programas desde una especificación a nivel de diseño.[GÓMEZ 2008]

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 cómodo. [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 emplea ya que es fácil para modelar y porque posibilita el trabajo en entorno libre.

RequisitePro es una herramienta centrada en documentos, que almacena los requisitos asociándolos a documentos (aunque también permite guardarlos directamente en la base de datos). Contribuye especialmente 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 [MCDONALD LANDAZURI 2005].

IRqA es una herramienta de la Ingeniería de Requisitos especialmente diseñada para soportar el proceso completo de dicha ingeniería. En IRqA el ciclo de especificación completo incluye la captura de requisitos, análisis, especificación de sistema, validación y la organización de requisitos es soportada por modelos estándares [MCDONALD LANDAZURI 2005].

QaTraq es un sistema de gestión de casos de prueba con capacidad de realizar reportes y seguimiento de los resultados de las pruebas. QaTraq está disponible como *Open Source*[8] Software. Está soportada para plataformas sobre Windows, Linux y Solaris [MCDONALD LANDAZURI 2005].

En el capítulo 2 se analizarán otras herramientas, las cuales están enmarcadas en el trabajo con las técnicas formales.

1.7 CONCLUSIONES PARCIALES

En los últimos años la industria de software en Cuba ha tenido un desarrollo vertiginoso. Es preocupación de todos los especialistas el tema de la calidad, específicamente la utilización de herramientas y técnicas que ayuden a identificar los posibles problemas, ineficiencias u oportunidades de mejora del software. Después de haber realizado un análisis de la información recopilada sobre temas tratados en el presente capítulo se puede determinar que para lograr la calidad e integridad necesaria en un producto software y además que los clientes queden satisfechos, es necesario que los productos estén libres de errores. Esto trae consigo que los desarrolladores tomen conciencia de lo importante que es la corrección del software y la influencia que tiene en los resultados finales de los productos para lograr competir en el mercado ya sea a nivel nacional o internacional. Además se arribó a la conclusión de que las técnicas formales son superiores con respecto a las tradicionales que son las que actualmente se utilizan en el proceso de desarrollo del software, por lo que en el próximo capítulo se analizarán herramientas basadas técnicas formales y que contribuyen a aumentar la corrección, eficiencia e integridad del software.

[8] Open Source: Herramienta libre.

CAPÍTULO 2: LA CORRECCIÓN, EFICIENCIA E INTEGRIDAD DEL SOFTWARE

2.1 INTRODUCCIÓN

Desarrollar un software es una labor compleja y requiere de mucho empeño, el personal encargado de esta tarea dedica gran esfuerzo para que los productos satisfagan las necesidades de los usuarios, para ello es importante que se apliquen correctamente elementos y características relacionadas con la calidad y que a su vez el software sea lo más óptimo posible.

Es importante destacar que la arquitectura, diseño e implementación de cada producto es de manera diferente sin embargo en su construcción intervienen factores de calidad que le dan la elegancia y perfección a dicho software, es imprescindible que estos factores se tengan en cuenta durante todas las etapas de desarrollo del software para evitar arrastrar errores a las últimas fases, lo cual traería consigo cambios en el mismo e innumerables gastos.

En ocasiones el software desarrollado en algunos proyectos llega con faltas graves a la etapa de pruebas, donde los errores son más costosos, y esto se debe en muchos de los casos a la falta de eficiencia, corrección e integridad del mismo.

Según lo anteriormente dicho en este capítulo se abordarán técnicas que formalmente garantizan corrección, eficiencia e integridad al software en todo su ciclo de vida, así como las herramientas basadas en dichas técnicas.

2.2 CORRECCIÓN DEL SOFTWARE

Es el factor de calidad externo fundamental. Es un factor muy fácil de definir, pero solo se puede verificar si las especificaciones son exhaustivas. Es la capacidad que tiene el producto software para realizar con exactitud todas sus tareas de forma correcta, tal y como se define en las especificaciones. Es la cualidad principal que debe tener un sistema bien desarrollado, ya que este debe cumplir con los requisitos y consideraciones planteados con anterioridad. Si un sistema no hace lo que se supone que debe hacer, poco importa cuán rápido es, si tiene bonita interfaz de usuario, si es amigable entre otras consideraciones que se hagan sobre él. El primer paso y el más difícil hacia la corrección es ser capaz de especificar de una forma precisa y con claridad los requisitos del sistema.

Según [S.PRESSMAN 1998] la corrección es hasta dónde satisface un programa su especificación y consigue los objetivos de la misión del cliente. Éste factor se subdivide en tres criterios que se muestran en [CERVERA PAZ 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.

Uno de los problemas de la corrección es que se presupone la confianza en los distintos componentes involucrados en la producción del sistema; compilador, bibliotecas, módulos, sistema operativo, entre otros.

Para garantizar la corrección, se emplean métodos usualmente condicionales, en la que cada nivel confía en la corrección de los inferiores para de esta forma asegurar también su correcto estado, 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.

Figura 5: Esquema de niveles de la corrección del software.

Se realiza a multinivel ya que un sistema de software importante, incluso uno pequeño según los estándares de hoy, implica a tantas áreas que sería imposible garantizar su corrección manejando todas las componentes y propiedades en un solo nivel.

A un programa le corresponde operar correctamente o suministrará poco valor a sus usuarios. La corrección es el grado en el que el software lleva a cabo una función requerida. La medida más común

de corrección son los defectos por KLDC[9], en donde un defecto se define como una falla verificada de conformidad con los requisitos.

Una práctica muy común en los desarrolladores es la optimización excesiva, lo importante es mantener un balance adecuado entre eficiencia y corrección.

2.3 EFICIENCIA DEL SOFTWARE

La **eficiencia** es uno de los factores más importantes que se tiene en cuenta cuando se mide la calidad del software. La misma 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 acorde con el período de tiempo establecido.

La eficiencia se define como el logro de los resultados con economía de los recursos, por lo tanto lleva implícito el concepto de eficacia que se refiere al logro de objetivos, y esto a la vez es una situación interesante puesto que con excepción de los casos en los que se define el objetivo de manera numérica no es posible establecer el grado de eficiencia con el cual se está desempeñando. Éste factor se subdivide en dos criterios que se muestran en [CERVERA PAZ 1997]:

Eficiencia en ejecución: Atributos del software que minimizan el tiempo de procesamiento.

Eficiencia en almacenamiento: Atributos del software que minimizan el espacio de almacenamiento necesario.

Según plantea [S.PRESSMAN 1998] la eficiencia es el conjunto de recursos informáticos y de código necesarios para que un programa realice su función.

[9] KLDC (acrónimo inglés de Kilo Lines Of Code, esto es, 1.000 líneas de código).

2.4 INTEGRIDAD DEL SOFTWARE

En esta época donde cada vez son más los intrusos informáticos y los virus, la integridad del software juega un rol muy importante tanto en el desarrollo y la construcción de cualquier tipo de software como en la utilización del producto luego de su implantación. Este atributo mide la habilidad de un sistema para soportar ataques (tanto accidentales como intencionados) contra su seguridad. El ataque se puede ejecutar en cualquiera de los tres componentes del software, ya sea en los programas, datos o documentos.

[S.PRESSMAN 1998] califica la integridad como el grado hasta dónde se puede controlar el acceso al software o a los datos por individuos no autorizados. Éste factor se subdivide en tres criterios que se muestran en [CERVERA PAZ 1997]:

- **Control de accesos:** Atributos del software que proporcionan control de acceso al software y a los datos que maneja.
- **Facilidad de auditoría:** Atributos del software que facilitan la auditoría de los accesos al software.
- **Seguridad:** La disponibilidad de mecanismos que controlen o protejan los programas o los datos.

2.5 TÉCNICAS FORMALES UTILIZADAS EN EL PROCESO DE DESARROLLO DEL SOFTWARE

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.Los métodos formales se basan en el empleo de técnicas, lenguajes y herramientas definidos matemáticamente para cumplir objetivos tales como facilitar el análisis y construcción de sistemas confiables independientemente de su complejidad, delatando posibles inconsistencias o ambigüedades que de otra forma podrían pasar inadvertidas. Las técnicas formales permiten crear una especificación sin ambigüedades que sea más completa y constante que la que se utilizan en los otros métodos. [FRANCISCO JOSÉ GALÁN MORILLO]

En los últimos años la idea de que la formalización matemática del software es el enfoque más apropiado para conseguir mejorar su calidad va adquiriendo cada vez más fuerza. Los partidarios de

los métodos formales defienden que su empleo, a lo largo de todo el ciclo de vida (ver anexo 4), facilita el desarrollo de especificaciones claras, concisas y no ambiguas, permite el análisis funcional de la especificación y posibilita el desarrollo de implementaciones correctas respecto a su especificación. Sin embargo otros aseguran que el empleo de métodos formales supone un volumen de trabajo considerable; un aumento en los costes y tiempo de desarrollo; y que debe quedar supeditado a herramientas que lo automaticen.

Trabajar con técnicas formales no significa desplazar o sustituir la forma tradicional de ingeniería, ya que estos métodos se pueden integrar favorablemente en el proceso de desarrollo del software. Normalmente la construcción de 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.

La aplicación práctica de dichos métodos formales en la industria del software encuentra una gran resistencia, debido fundamentalmente a dos deficiencias: la falta de herramientas prácticas que demuestren sus ventajas teóricas; y la necesidad de un nuevo proceso de adaptación en los usuarios y diseñadores, que no están acostumbrados a la utilización de este tipo de técnicas, ya que si no se conoce su correcto funcionamiento podría atrasar el proyecto ya sea por no saber como emplearlos o simplemente por no seleccionar la más conveniente dada la situación.[DUQUE 2000]

Otras causas para no usar técnicas formales en la industria del software es que gran parte de los ingenieros software, especialmente los que cuentan con mayor experiencia, no han sido entrenados en el uso de estas técnicas que requieren conocer conceptos de matemática discreta y de lógica; las técnicas de descripción formal no son adecuadas para la especificación de algunos sistemas software, especialmente interfaces de usuario.

Tras el estudio de estas técnicas vale destacar que a pesar de que con su empleo se comprende mejor el sistema y que la comunicación con el cliente es más sencilla dada la claridad de las descripciones de los requisitos del usuario; estos métodos no pueden solucionar todos los problemas que se presenten ya que la calidad de un producto va mas allá de la utilización de estas técnicas. Sin embargo, son muy provechosas sus ventajas cuando quiere lograrse un poco más de calidad de la que tradicionalmente se obtiene. [GIL 1999]

2.5.1 Clasificación de los métodos formales

El panorama actual en el campo de los métodos formales dista mucho de la homogeneidad. Hoy día existe gran variedad de métodos y técnicas de especificación formal, de diversas características, ámbito de aplicación y experiencia publicada. En general, lo que más predomina en este campo son las técnicas de especificación, notaciones formales que favorecen la descripción precisa de sistemas y propiedades. Más escasos resultan los métodos, vistos estos como procedimientos sistemáticos que sirven de guía al diseñador, en todo o parte del proceso de desarrollo. Esta falta de sistematización implica generalmente la necesidad o conveniencia de cierta experiencia en el desarrollo de abstracciones y la modelación de sistemas. Los criterios de clasificación también son muy múltiples y de distinta relevancia en función del enfoque adoptado. Desde el punto de vista de los aspectos que modelan el sistema se pueden diferenciar según [GIL 1999] entre:

Métodos que modelan la funcionalidad del sistema: Son la mayor parte.
Métodos orientados a la descripción de propiedades de concurrencia: Muy apropiados para la descripción de protocolos de comunicación y sistemas distribuidos.

Métodos que permiten modelar el tiempo: Aunque menos desarrollados que los anteriores son de gran importancia en la aplicación de sistemas de tiempo real.

La fase de desarrollo del sistema también es importante para valorar la aplicabilidad de un método formal. Se pueden dividir en [GIL 1999]:

Técnicas constructivas: Describen directamente la funcionalidad del sistema y suelen tener como resultado una especificación formal que puede ser fácilmente ejecutable. Son, por tanto, muy apropiadas para realizar un prototipado rápido del sistema en fases intermedias o finales. Su principal inconveniente suele ser la imposibilidad para especificar de forma explicita las propiedades del sistema, lo que dificulta la verificación de propiedades.

Técnicas no constructivas: Describen el sistema a través de sus propiedades, por lo que también suelen llamarse técnicas orientadas a propiedades. Facilitan la verificación de propiedades, su comprobación de consistencia, entre otras. Sus inconvenientes radican en la imposibilidad de describir la estructura del sistema y la dificultad para decidir si una especificación es o no completa. Por ello, son más recomendables para la fase inicial de captura y análisis de requisitos del sistema.

Sin embargo, la clasificación más común se realiza sobre la base del modelo matemático subyacente en cada método, de esta manera podrían clasificarse en:

Especificaciones basadas en lógica de primer orden y teoría de conjuntos: Permiten especificar el sistema mediante un concepto formal de estados y operaciones sobre estados. Los datos y relaciones/funciones se describen en detalle y sus propiedades se expresan en lógica de primer orden. La semántica de los lenguajes está basada en la teoría de conjuntos. [BOTON 2008] Una característica común de muchos de los métodos de este tipo es que permiten definir condiciones invariantes sobre el espacio de datos, mediante predicados sobre sus componentes. Los métodos de este tipo más conocido son: Z, VDM y B. [GIL 1999]

Especificaciones algebraicas: Proponen una descripción de estructuras de datos estableciendo tipos y operaciones sobre esos tipos. Para cada tipo se define un conjunto de valores y operaciones sobre dichos valores. Las operaciones de un tipo se definen a través de un conjunto de axiomas o ecuaciones que especifican las restricciones que deben satisfacer las operaciones. Métodos más conocidos: Larch, OBJ, TADs.

Especificación de comportamiento:

Métodos basados en álgebra de procesos: Modelan la interacción entre procesos concurrentes. Esto ha potenciado su difusión en la especificación de sistemas de comunicación (protocolos y servicios de telecomunicaciones) y de sistemas distribuidos y concurrentes. Los más conocidos son: CCS, CSP y LOTOS. [BOTON 2008], [GIL 1999].

Métodos basados en Redes de Petri: Una red de Petri es un formalismo basado en autómatas, es decir, un modelo formal basado en flujos de información. Permiten expresar eventos concurrentes. Los formalismos basados en redes de Petri establecen la noción de estado de un sistema mediante lugares que pueden contener marcas. Un conjunto de transiciones (con pre y post condiciones) describe la evolución del sistema entendida como la producción y consumo de marcas en varios puntos de la red [BOTON 2008].

Métodos basados en lógica temporal: Diferentes versiones de la lógica temporal han sido usadas satisfactoriamente para especificar sistemas, especialmente sistemas de tiempo real. Estos métodos nos permiten especificar de forma explícita las propiedades de seguridad y viveza de un sistema. Un

sistema se especifica a través de un conjunto de fórmulas lógicas que definen relaciones y sucesos que ocurren en el tiempo. Además, permiten trabajar con especificaciones parciales del sistema. Su principal inconveniente radica en su incapacidad para expresar la estructura de un sistema. Por ello, su utilización suele limitarse a las primeras fases del diseño [GIL 1999]. Se usan para especificar sistemas concurrentes y reactivos. Los sistemas reactivos son aquellos que mantienen una continua interacción con su entorno respondiendo a los estímulos externos y produciendo salidas en respuestas a los mismos, por lo tanto el orden de los eventos en el sistema no es predecible y su ejecución no tiene por qué terminar. Una especificación escrita en lógica temporal describe las secuencias admisibles de estado (incluyendo estados concurrentes) para el sistema especificado [BOTON 2008] .

2.5.2 Técnicas de Descripción Formal

Los métodos de especificación formal, son métodos formales dedicados a escribir especificaciones de sistemas. La mayor parte de los métodos de especificación formal son notaciones que permiten especificar y razonar sobre el comportamiento de sistemas. Esta característica hace necesaria la experiencia y adecuada formación del equipo de desarrollo para trabajar con abstracciones y modelos de sistemas.[DUQUE 2000]

Una especificación formal es una expresión matemática que contiene la descripción de un sistema. Una característica deseable, y en algunos casos necesaria en la especificación de normas, es que la especificación sea independiente de la implementación; es decir, los detalles de implementación deben incorporarse en fases posteriores. [MUÑOZ 2008]

En la actualidad, existe un amplio consenso en considerar las técnicas de descripción formal[10], como una forma adecuada de describir el comportamiento de sistemas reactivos de procesamiento de la información. Las principales mejoras que se persiguen con el empleo de las TDF son: corrección del sistema construido respecto a una especificación formal; mejora de la calidad del sistema (entendiendo calidad como el grado de cumplimiento de los requisitos y expectativas planteadas); y aumento de la productividad.

Las características deseables de una TDF dependen de la fase del proceso de desarrollo en la que ésta se aplique [DUQUE 2000]:

[10] Técnicas de Descripción Formal (TDF)

En la fase inicial, de captura y especificación de requisitos, es preferible obtener una especificación flexible del sistema en función de sus propiedades más importantes.

En la fase de diseño de la arquitectura es necesario poder describir los componentes del sistema, su interfaz y como interactúan entre ellos. Es decir, es necesario poder expresar la estructura del sistema especificado.

En la fase de implementación los componentes tienen que describirse de forma que su comportamiento pueda ser generado por máquinas, y por tanto, es necesario utilizar un lenguaje de programación. El proceso de traducción de un lenguaje de especificación a un lenguaje de implementación se puede automatizar en gran medida. Por tanto, si se emplean TDF, podemos decir que esta fase es semiautomática.

Verificación Formal

Las TDF contribuyen a reducir el riesgo inherente a la fase de captura y especificación de requisitos del sistema. No obstante, es importante ser consciente de que una especificación de un sistema escrita en una notación formal puede contener también errores. Para aumentar la confianza del diseñador en la corrección del método es preciso realizar operaciones de verificación sobre la especificación. Muchas veces esta verificación se realiza de manera informal a través de inspecciones y reuniones de revisión. Sin embargo, la base matemática de los métodos de especificación formal posibilita la verificación formal en dos aspectos: verificación formal de propiedades formuladas sobre la especificación, y verificación formal de que la implementación satisface la especificación. [DUQUE 2000]

Una de las más importantes motivaciones que impulsó el desarrollo de las TDF fue la posibilidad de realizar una verificación formal del sistema obtenido. No obstante, es en este último punto donde surgen más controversias sobre la utilidad de las mismas. Por una parte, la experiencia ha demostrado que la verificación formal de grandes especificaciones es, en muchos casos, inviable por el alto coste de recursos que es necesario invertir. Por lo que entonces es necesario limitar o concentrar las verificaciones a partes del sistema que por su naturaleza se consideren más críticas.

Un método alternativo [DUQUE 2000] a la verificación formal citada –verificación formal de que la implementación satisface la especificación-, lo constituye un proceso de refinamiento iterativo donde la especificación formal de un sistema se obtiene tras la aplicación sucesiva de un conjunto de simples

refinamientos o transformaciones aplicadas a una especificación inicial del sistema con un elevado nivel de abstracción. Cada una de las fases de este proceso toma como entrada la especificación resultante del refinamiento anterior y produce como salida otra especificación con mayor cantidad de detalles de implementación. Éste proceso continúa hasta que se dispone de una especificación con el nivel de detalle suficiente como para poder traducirla a un lenguaje de programación convencional. Las especificaciones de entrada y salida de un refinamiento se pueden relacionar mediante relaciones de equivalencia o de orden. Las ventajas, a priori, de este método es que las verificaciones que se hacen son mucho más simples.

2.5.3 Técnicas formales que garantizan corrección, eficiencia e integridad

Las técnicas formales que garantizan corrección, eficiencia e integridad al software se basan en lenguajes formales para su aplicación. Estos lenguajes describen formalmente los requisitos de un sistema, son el extremo opuesto al lenguaje natural[11], han sido aplicados en el mundo de la Ingeniería de Requisitos desde hace años. Sin embargo, resultan muy complejas en su utilización y para ser entendidas por el cliente.

El mayor inconveniente es que no favorecen la comunicación entre cliente y analista. Por el contrario, 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, ya que en éstos toda ambigüedad es eliminada. [ESCALONA 2002]

Las características que tienen estos lenguajes son las siguientes:

- ✓ Se desarrollan de una teoría preestablecida.
- ✓ Componente semántico mínimo.
- ✓ Se han desarrollado como un medio para formalizar matemáticamente una teoría.
- ✓ Su sintaxis produce oraciones no ambiguas, rigurosamente definidas.
- ✓ La importancia del rol de los números.
- ✓ Constituyen un poderoso instrumento para la investigación y el procesamiento del lenguaje natural por computadora.

[11] Es el medio que utilizamos de manera cotidiana para establecer nuestra comunicación con las demás personas

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

2.5.3.1 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].

En este proceso de forma general pueden utilizarse las **técnicas no constructivas**, o también **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 la modelación. 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.

El proceso comienza con la realización de la captura de requisitos, el grupo de técnicos toma la información suministrada por los usuarios y clientes. Esta información puede provenir de fuentes muy diversas: documentos, aplicaciones existentes, a través de entrevistas, entre otras. Sobre la base de ella, el equipo de desarrollo elabora el catálogo de requisitos. Finalmente con la validación de requisitos se realiza la valoración de los mismos, comprobando si existen inconsistencias, errores o si faltan requisitos por definir [ESCALONA 2002].

El proceso de especificación de requisitos se puede dividir en tres grandes actividades:

1- **Captura de requisitos:** Cuando se cuenta con un personal experimentado, esta actividad resulta más sencilla, puesto que se distinguen oportunamente los métodos a seguir y se selecciona 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.

   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 ésto no puede ser automatizado. [POLLAN CAÑETE 2007]

Existen herramientas en las cuales pueden ser modelados, algunos ejemplos:

- El Rational Rose (en entorno Windows).
- Visual Paradigm Suite (en entorno libre).

2- **Definición de requisitos:** Es el proceso mediante el cual se representan y se describen explícitamente los requisitos capturados. Una vez que han sido definidos los requisitos pueden atribuírsele algunas técnicas que ayuden a un mayor entendimiento de los mismos y a acelerar el proceso de desarrollo.

3- **Validación de requisitos:** Una vez que se definen los requisitos, estos necesitan ser validados, o sea, demostrar que los requerimientos obtenidos estén correctos y define realmente el sistema que el usuario necesita. En términos tradicionales no son muchos los métodos conocidos de validación de requisitos, éstos consisten en revisar la información obtenida en la definición de requisitos, que por lo general se hace manualmente. [POLLAN CAÑETE 2007]

No existe una única técnica estandarizada que garantice la plena calidad en la realización de este proceso. A continuación se presentan un grupo de técnicas que de forma clásica han sido utilizadas para esta actividad en el proceso de desarrollo de todo tipo de software.

Larch: Es un proyecto de investigación, que según manifiestan [DUQUE 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[12] y LSL[13]. 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

[12] BISL (Lenguaje de Interfaz Larch, o en inglés, Behavioral Interface Specification Language)
[13] LSL (Larch Shared Language)

el vocabulario matemático usado en especificaciones de pre y post condiciones, sin importar el lenguaje [DUQUE 2000] y [LARCH 2001].

SDL[14] [DUQUE 2000], [GIL 1999] es un lenguaje de descripción formal, de propósito general, normalizado por ITU[15].Incluye elementos para definir la estructura, el comportamiento y los datos del sistema, permitiendo especificar características temporales e indeterminismo. Incorpora dos sistemas de representación semánticamente equivalentes: uno gráfico (SDL/GR) y otro textual (SDL/PR). Existen múltiples herramientas, tanto comerciales como de dominio público basadas en esta técnica, entre ellas se destacan:

QUEST: Un entorno para la descripción e integración de subsistemas, el análisis de requisitos, la visualización de comportamiento y el estudio de rendimientos.

ProcGe: Es una herramienta comercial de desarrollo de sistemas por refinamientos sucesivos que permite generar código de forma automática.

Con la ayuda de SDL se han realizado numerosos e importantes proyectos como es el caso de INSYDE, proyecto Sprit para el estudio de diseños híbridos hardware-software, en el cual Alcatel utilizó SDL para la especificación del software de un sistema de vídeo bajo demanda. Otro ejemplo del uso de SDL es en NewCore, sistema de telecomunicaciones formalmente verificado en SDL.

En [SÁNCHEZ 2002] se presenta el método **VRU**[16] que permite validar requerimientos de usuarios de forma 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.

OASIS: Es un lenguaje de especificación formal orientado a objetos, desarrollado en el seno del grupo, y los modelos gráficos de análisis propuestos en la metodología, eliminando la ambigüedad de éstos y poniendo en práctica las interesantes propiedades de los lenguajes de especificación formales.

Una de las contribuciones más relevantes de OO-Method es que se genera de forma automática la especificación del sistema en OASIS[17] a partir de los modelos gráficos obtenidos en la fase de análisis

[14] SDL (Specification and Description Language)
[15] ITU (International Telecommunication Union)
[16] VRU (Validación de Requisitos de Usuario)
[17] OASIS (Open and Active Specification of Information Systems)

y dicha especificación puede servir como documentación del sistema. Por último, diversos generadores de código nos permiten obtener implantaciones de la especificación en distintos entornos de desarrollo de software declarativo e imperativo. [FRANCISCO JOSÉ GALÁN MORILLO]

Un ejemplo interesante donde es observable la experiencia en la Ingeniería de Requisitos con el lenguaje de especificación formal orientado a objetos OASIS es a través del análisis y diseño de un microscopio efecto-túnel, que mediante esta forma de especificación se introdujeron mejoras de diseño en el sistema.

En esta etapa de desarrollo es fundamental que sus actividades se hagan lo más precisas posible, ya que en esta fase se utilizan técnicas formales orientadas a la definición y validación correcta de los requerimientos, de forma tal que al pasar a la siguiente etapa, sean menos los errores y sea más fácil de modelar el sistema. Estas técnicas formales son de suma importancia ya que las mismas mejoran la integridad y la seguridad de los sistemas, contribuyendo a la corrección y eficiencia del software, y de una forma u otra, influyendo en los otros factores de calidad.

Esto quiere decir que para obtener un producto fiable, íntegro y eficiente lo primero que hay que garantizar es la especificación de requisitos, si puede hacerse formalmente, mucho mejor.

2.5.3.2 Técnicas Formales para la Modelación del sistema

Cuando se modela el sistema el mayor esfuerzo se le dedica al desarrollo de la arquitectura del sistema que posteriormente será construido, descomponiendo el sistema en módulos o componentes. Además se realizan las primeras actividades de la implementación que tienen más relevancia; se toma como entrada la especificación del sistema obtenida en la fase anterior y concluye cuando se dispone de una detallada documentación de cada uno de los módulos, el mecanismo de comunicación entre los componentes del sistema, los algoritmos y las estructuras de datos.

Ésta fase suele dividirse en dos partes: en la primera se obtiene un diseño de alto nivel de cada uno de los componentes o módulos del sistema; y en una segunda parte, se refinan dichos módulos hasta obtener un diseño con el nivel de detalle suficiente para pasar a la siguiente fase de implementación. [DUQUE 2000]

Llegado el momento para la realización del diseño del sistema, es aconsejable el uso de métodos que formalmente establecidos, puedan darle la perfección a la arquitectura que se quiere. Para esto es

necesaria la utilización de **técnicas constructivas**, que son TDF para realizar un modelado rápido del sistema final. Pero, no se puede especificar las propiedades del sistema de forma explícita, lo que implica que no puedan comprobarse explícitamente algunas propiedades. Algunas técnicas de éste tipo son: LOTOS[18] y CSP[19]. [DUQUE 2000]

LOTOS: En español, Lenguaje de Especificación de Orden Temporal. Es una técnica de descripción formal según [GIL 1999] normalizada por ISO en el año 1989, con el objetivo de expresar formalmente normas de servicios y protocolos OSI. El mecanismo básico que emplea este lenguaje para la especificación de un sistema consiste en la descripción de la relación temporal entre las posibles interacciones que constituyen el comportamiento externamente observable de dicho sistema. En esta definición se resalta el carácter de descripción extensional de las especificaciones LOTOS; uno de los objetivos fundamentales del lenguaje es permitir que el especificador defina cómo tiene el sistema que relacionarse con su entorno, tratando de obviar, en lo posible, la estructuración interna del sistema que permite ese comportamiento.

El condicionar lo menos posible la arquitectura de futuras implementaciones es una característica importante que deben perseguir las descripciones LOTOS. Una especificación debe contener las mínimas restricciones posibles a la estructura que el implementador quiera dar a la realización del sistema. Los constructores del lenguaje ciertamente favorecen la consecución de ese objetivo.

El lenguaje LOTOS está formado por dos sublenguajes que se complementan para obtener la descripción más adecuada de un sistema. La parte que trata el comportamiento y el control de las interacciones entre procesos está principalmente basada en el álgebra de procesos *Calculus of Communicating Systems* (CCS), aunque con fuertes influencias de álgebras de procesos similares como CIRCAL y fundamentalmente el CSP.

Por otro lado, tiene un segundo componente que se ocupa de la descripción de las estructuras de datos y expresiones de valor. Éste sublenguaje de tratamiento de datos está basado en la teoría formal de tipos abstractos de datos. Éste segundo componente es independiente del que trata de la descripción de los comportamientos y por tanto, fácilmente sustituible por otro de similar funcionalidad.

[18] LOTOS (Language of Temporal Ordering Specifications)
[19] CSP (Communicating Sequential Processes)

Se utilizó LOTOS dentro de la implementación del proyecto SEDOS. Otra referencia interesante donde se usa el LOTOS junto con el lenguaje META-IV es para programar robots off-line.

La fuerte base matemática que sustenta la semántica formal de LOTOS hace posible el desarrollo y aplicación de métodos de validación y verificación de propiedades y equivalencias. Ello es de vital importancia para lograr el objetivo prioritario de obtener sistemas altamente fiables, íntegros y eficientes.

CSP: Es un lenguaje para describir patrones de interacción en sistemas concurrentes, el cual es soportado por una teoría matemática bien fuerte, un conjunto de herramientas de prueba y literatura extensiva. Permite describir un sistema a través de un conjunto de componentes (procesos) que operan independientemente y que se comunican entre ellos a través de canales definidos para tal propósito (la restricción de que los procesos componentes fueran secuenciales se eliminó entre 1978 y 1985, si bien se mantuvo el nombre del lenguaje).Como se ha dicho anteriormente, esta técnica entra en la clasificación de las constructivas-algebraicas. Dado estos fundamentos con el empleo de esta técnica se garantiza la corrección del software, ya que las herramientas de prueba contribuyen a la eliminación de inconsistencias y errores, de esta forma también de garantiza la eficiencia del producto.

Este lenguaje fue definido en un artículo de 1978 por C.A.R. Hoare, cuyo refinamiento dio origen a una notación más flexible en 1985 [DUQUE 2000]. Ha sido aplicado en la industria del software extranjera para verificar y especificar aspectos concurrentes en varios sistemas. [POLLAN CAÑETE 2007]

El modelo de comunicación de CSP, en el que la información se transmite solamente a través de canales nombrados explícitamente, es uno de los principios básicos del lenguaje de programación OCCAM.

Las herramientas más conocidas que implementan CSP son **FDR**, **ProBE** y **Casper**.

FSP[20] : Es una variante simple de CSP. La semántica de los procesos FSP está dada en términos de sistemas de transición de estados y trazas, al igual que para CSP. En particular, los procesos (definidos en forma textual en CSP) pueden verse gráficamente como sistemas de transición de estados con el uso de la herramienta asociada LTSA (*Labelled Transition System Analyser*). Esta técnica garantiza corrección del software. [LONDON 2007]

[20] FSP (Finite State Processes)

CCS[21]**:** Fue definido por Milner en 1980. CCS fue la primera álgebra para describir concurrencia y comunicación entre procesos. La semántica de CCS se define a través de sistemas de transiciones etiquetados y estos sistemas son los modelos naturales de las lógicas temporales y modales.

Entre las herramientas más importantes que emplean CCS destacan el **Concurrence Workbench** de la Universidad de Edimburgo, **JACK** del IEI-CNR y **ATG FC-TOOLS** del INRIA/CMA-ENSMP.

Z: Es una notación formal basada en lógica de predicados de primer orden y en teoría de conjuntos. Se comenzó a desarrollar en la Universidad de Oxford en el año 1980 desarrollado por miembros del Laboratorio de Computación de dicha universidad desde los años 70. Actualmente, está en proceso de normalización por parte de ISO.

Está orientada a la construcción de modelos de forma incremental. Se utiliza para describir software. La utilización de la lógica de predicados permite la descripción abstracta del efecto de cada operación en un sistema y de los invariantes que deben cumplirse, así como el razonamiento sobre el comportamiento del sistema. Por otra parte, la teoría de conjuntos permite modelar el sistema mediante entidades matemáticas conocidas, tales como conjuntos, relaciones, funciones y secuencias. De ahí deviene la importancia de esta técnica para la integridad del software, ya que la misma no permite ambigüedades y ayuda a la verificación del software. [GIL 1999]

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). [POLLAN CAÑETE 2007]

Z incluye una notación de esquemas para ayudar a la estructuración de especificaciones. Su propia forma de representación hace que el sistema pueda ser verificado formalmente por otras técnicas que están destinadas a ello. Sin embargo no es buena para todo, por ejemplo para la concurrencia no es conveniente su uso [POLLAN CAÑETE 2007].

Varios casos de estudios referidos en [GIL 1999] demuestran la importancia requerida del uso de esta técnica en el desarrollo de múltiples sistemas reales de gran tamaño. A continuación estos ejemplos:

[21] CCS (Calculus of Communicating Systems)

Entre la Universidad de Oxford y los laboratorios Hursley de IBM presentaron un proyecto llamado CICS, el mismo consta en desarrollar un sistema en tiempo real.

El proyecto CIDS, el cual verifica parte del software del Airbus A330/340 mediante métodos formales, especialmente Z.

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. [POLLAN CAÑETE 2007]

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 [GOGUEN 2005] , lo presenta como una familia de OBJ. 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. [POLLAN CAÑETE 2007]

VDM[22] : [GIL 1999] Es una colección de técnicas formales para la descripción y el desarrollo de sistemas software. Tiene su origen en trabajos llevados a cabo a mediados de los setenta por Cliff Jones y Dines Bjørner en un laboratorio de IBM en Viena. Utiliza una notación matemática precisa para determinar la funcionalidad que debe proporcionar el sistema.

VDM consiste en un lenguaje de especificación denominado **VDM-SL[23]**. Está formado por reglas para el refinamiento de datos, operaciones que permiten establecer vínculos entre requisitos abstractos y especificaciones detalladas, y una teoría de pruebas para verificar propiedades y decisiones de diseño.

Las especificaciones VDM se construyen en base a modelos de un estado subyacente del sistema, el cual puede ser modificado mediante operaciones definidas a través de pre y postcondiciones.

[22] VDM (Vienna Development Method)
[23] VDM-SL (VDM Specification Language).

VDM-SL se basa en la utilización de una lógica de funciones parciales y, al igual que Z, en teoría de conjuntos con secuencias y funciones. A diferencia de Z, en VDM-SL se describen de forma explícita las precondiciones y postcondiciones asociadas a una operación.

Entre los trabajos más importantes en los que se ha utilizado VDM se encuentran:

CDIS, proyecto presentado en 1992 por la empresa Praxis, en el cual se emplea VDM para verificar parte del mecanismo de representación de información del sistema de control de tráfico aéreo de Londres.

En ACS se especificaron mediante VDM los requisitos de seguridad de un sistema de almacenamiento de explosivos del ejército británico. [DUQUE 2000]

Otro ejemplo en el cual ha servido VDM durante los más de treinta años de vida están: [BJORNER 2006], [POLLAN CAÑETE 2007].

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

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 LARSEN 2006]

Estelle: Fue desarrollado para la especificación de servicios y protocolos del modelo de referencia OSI de ISO. No obstante, es útil para la especificación de sistemas distribuidos en general. 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. Está basado en máquinas de estados finitos que se comunican entre sí mediante colas y variables globales. Su sintaxis es muy similar a la del lenguaje PASCAL. Ha sido aplicado a sistemas tanto Unix como Windows, por esto puede utilizarse con gran utilidad en la esfera del software libre. [GIL 1999], [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. Éste producto fue

desarrollado para construir una plataforma de prueba, evaluación del funcionamiento y la optimización de la implementación de los métodos para Estelle. [TENNEY 2000]

La ventaja de este tipo de lenguajes es que las especificaciones de los sistemas son muy naturales, ya que las descripciones se basan en conceptos muy conocidos. Además, proporcionan al equipo de implementación del sistema una guía de trabajo [GIL 1999]. Existen herramientas que traducen automáticamente ESTELLE a C++ como es el caso de PetDingo, la cual fue desarrollada por NIST, puede encontrarse una versión en [TENNEY 2000].

Otro ejemplo de herramientas para la traducción automática de ESTELLE pero en este caso a C es EDT, la cual incluye simuladores y ayudas de depuración. Mediante ESTELLE se han especificado y verificado varios protocolos ISO, por ejemplo, ROSE [GIL 1999].

B: Como parte de los métodos formales derivados de Z, ha sido desarrollada igualmente por Jean Raymond Abrial, con el objetivo de especificar y verificar tanto sistemas hardware como software. Se trata de un conjunto de técnicas matemáticas apropiadas generalmente para el diseño e implementación de componentes software. Los sistemas se modelan como una colección de máquinas abstractas interdependientes que se describen con un enfoque orientado a objetos mediante la notación AMN (*Abstract Machine Notation*) [GIL 1999], [DUQUE 2000].

El Método B establece guías para la estructuración de grandes diseños y la comprobación de la consistencia de éstos. Teniendo en cuenta estos argumentos, se garantiza corrección y eficiencia en el sistema, permitiendo no dar paso a errores ni ambigüedades [GIL 1999].

Incluye un grupo de herramientas como **B-Toolkit** y **Atelier-B** que han sido aplicadas en varias ramas de la industria [DUQUE 2000].

Varios trabajos prácticos pueden encontrarse en [B 2007] como muestra de la Séptima Conferencia del Método B que tuvo lugar del 17 al 19 de junio del 2007 en Francia.

Petri Nets: Esta técnica toma el nombre en honor a su creador Karl Adam Petri, quien en 1962 creó lo que se conocen en español como Redes de Petri (RP). En varios artículos visitados se habla de una herramienta para el modelado de sistema, pero no es así, pues está demostrado que son métodos que contribuyen a modelar el comportamiento y la estructura de sistemas basados en la teoría de autómatas y llevar el modelo a condiciones límite, que en un sistema real son difíciles de lograr o muy

costosas. Su característica más relevante es que pueden ser analizadas de manera formal y obtener información del comportamiento dinámico del sistema modelado [GONZALES URMACHEA?] .

Es aplicable a modelos de redes abstractas, procesamientos paralelos y distribuidos, teoría de grafos, problemas de transporte, problemas de decisión y reconocimiento de patrones, entre otras [POLLAN CAÑETE 2007].Cuentan con un conjunto de herramientas disponibles que favorecen el trabajo de los desarrolladores debido a las ventajas que poseen, por lo que son muy conocidas y aplicadas.

Podemos ver aplicaciones prácticas de las RP en [SÁNCHEZ PALMA 2002], donde se definen las bases teóricas y prácticas necesarias para la validación automática de especificaciones del lenguaje OASIS, como parte de un proyecto de Comisión Interministerial de Ciencia y Tecnología (CICYT).

Larch: Para la modelación 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.

¿Por qué emplear en esta etapa estas técnicas?

En los últimos años la investigación sobre métodos formales crece en interés en el medio académico e industrial. Los lenguajes de especificación son cada vez más cercanos a los lenguajes de programación, las técnicas se acomodan mejor a los nuevos paradigmas de desarrollo (orientación por objetos o computación distribuida son ejemplos notables) y herramientas comerciales de calidad que soportan dichos métodos se consiguen en el mercado. A medida que los sistemas informáticos crecen en complejidad, las perdidas causadas por fallas en dichos sistemas son cada vez mayores. Cuando corrección se traduce en dinero, los métodos formales atraen a la industria [MUÑOZ 2008].

Muchas veces los métodos formales se implantan en lugares donde los costos por fallos en el sistema son mayores que el costo que se estima para la producción del software en general, ya que se espera a que estás cosas sucedan para ponerlos en práctica, ya sea por una u otra razón. No obstante, cualquier costo extra que se invierta en el proyecto en la capacitación del personal que usará éstas técnicas es insignificante en comparación con lo que puede ahorrarse con su empleo.

La aplicación de estas técnicas se ponen de manifiesto principalmente en actividades importantes como equipos médicos, ejemplo de ello son los marcapasos del corazón; en proyectos de aviación y monetarios, entre otros. A continuación se mencionan algunos ejemplos que demuestran fehacientemente la necesidad de su aplicación:

Entre 1985 y 1987 un software de control de equipamiento médico para radioterapia llamado Therac-25, por un error en el sistema ocasionó que seis personas estuvieran sobre expuestas a radiación.

En 1995 un profesor de matemáticas de la Universidad de Lynchburg descubre un error en los procesadores Pentium de la compañía Intel. Aunque Intel corrigió rápidamente el error y se comprometió a reemplazar los procesadores, el daño ya estaba hecho. Cerca de dos millones de procesadores defectuosos habían sido distribuidos alrededor del mundo. La falla fue un error de diseño en el algoritmo de división de punto flotante.

En 1996 la explosión de la nave espacial europea Ariane 5, 40 segundos después de su lanzamiento, fue un hecho que demostró que los sistemas informáticos no son infalibles. Los reportes oficiales indican que la perdida alcanzó los 500 millones de dólares. La comisión de expertos que investigó el desastre concluyó que la falla fue un número fuera de rango, que se produjo en un pedazo de código heredado del sistema de Ariane 4. Lo irónico del caso, es que este código no tenía ninguna utilidad en el nuevo sistema.

Como contrapartida a esto en [MACCOLL 1999] se muestran algunas aplicaciones de los métodos formales que son empelados y defendidos por desarrolladores en todo el mundo:

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. Ha sido reportado que las tasas de error fueron 36% mejores que las correspondientes a los promedios de la industria.

Se hace alusión al desarrollo de un sistema de información grande del control de tráfico aéreo utilizando una combinación de métodos formales. El sistema era grande (197,000 líneas de código C) con requerimientos significativos de tiempo real y de seguridad; las tasas de productividad global y las de error fueron comparables o mejores que los promedios de la industria.

Como se ha podido apreciar en cada ejemplo donde se aplicaron estas técnicas, con el empleo de las mismas se puede garantizar corrección, eficiencia e integridad, utilizando herramientas adecuadas para ello. En la etapa de modelación del sistema estas técnicas posibilitan que se determine la arquitectura del sistema formalmente, lo que contribuye a optimizar la calidad en la futura construcción del software.

2.5.3.3 Técnicas formales para la Construcción del sistema

Luego que se determina la arquitectura software definida en la fase anterior se construyen los componentes del sistema, de forma que puedan ejecutarse en el hardware seleccionado [DUQUE 2000].

Los programadores en esta fase son los máximos encargados de que el software funcione correctamente, ya que es en esta actividad donde realizan un mayor esfuerzo de trabajo. Para ello utilizan técnicas tradicionales que no siempre suelen ser las más adecuadas, sin embargo con el transcurso de los años y las experiencias de quienes desarrollan software ha quedado demostrado que cuando se aplican formalismos a un sistema se trabaja en busca de obtener la calidad deseada y esto contribuye a garantizar la corrección, la eficiencia e integridad al software.

Las técnicas formales que atestiguan lo anteriormente dicho son:

El término **interpretación abstracta** significa que al programa se le da una semántica distinta de la estándar, se basa en la semántica de un lenguaje de programación para definir análisis estáticos de programas que, por su construcción, son correctos .Su ejecución requiere una importante base matemática. Puede ser más o menos preciso, según el nivel de observación elegido. Si se observa sin mucho detalle, se puede realizar una abstracción de su semántica que es computable, aún siendo menos precisa que la original. Esto puede ser útil en cualquier etapa de desarrollo del software. [CERNUDA and MANUEL. 2001] [GALLARDO 2006]

Tiene como objetivo extraer información segura/correcta sobre el comportamiento dinámico de los programas sin necesidad de ejecutarlos para todas sus entradas y hacer esto automáticamente.

Entre las ventajas de usar valores abstractos se destacan:

La computabilidad: La posibilidad de asegurar que los resultados del análisis se obtienen en tiempo finito.

La generalización: 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]

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

Análisis Estático: Se basa en la interpretación abstracta, dada la semántica original de un programa, derivar otra semántica aproximada y computable. De este modo, 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. [POLLAN CAÑETE 2007]

Etapas del Análisis Estático

Para realizar análisis estático se tienen en cuenta las siguientes etapas: [POLLAN CAÑETE 2007] [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.
- **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. Buena parte de las aplicaciones prácticas de análisis estático están encaminadas a la optimización de código en compiladores.

En [CERNUDA and MANUEL. 2001] se hace una reflexión a que en ocasiones, lo que se desea verificar en un sistema no es su comportamiento a un nivel algorítmico riguroso. Frecuentemente, los programadores o diseñadores tienen un conocimiento sobre el sistema que podría ser aprovechado sin un análisis estático; es evidente que muchos programadores son capaces de afirmar que un programa dado contiene un error o que se bloqueará, aunque no sean capaces de demostrarlo algorítmicamente. Si todo el conocimiento que programadores, diseñadores, fabricantes, entre otros, tienen sobre los componentes software pudiera describirse formalmente y de manera sencilla, sería posible generar una "base de conocimientos" que describe todo lo que se sabe sobre el sistema que se está construyendo; esto permitiría verificar si los requisitos de ciertos componentes se cumplen o no y por tanto prever defectos de funcionamiento.

Uso de estas técnicas

La aplicación de estos métodos se ha difundido mundialmente, principalmente en el Reino Unido y en países desarrollados, que es donde mayormente son empleados, ya que ven las técnicas formales como una buena alternativa de desarrollo y de perfección en el software que se produce. Incluso en el programa de centros universitarios se ha insertado su estudio como una asignatura más de la carrera, dada su vital importancia y lo que representa para su economía un mal uso de técnicas tradicionales.

2.5.3.4 Técnicas formales para Prueba

Las pruebas es una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos específicos, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente.

El objetivo de esta etapa es encontrar y documentar los defectos que puedan afectar la calidad del software. Además se valida que el software trabaje como fue diseñado, se prueban los requisitos que debe cumplir el sistema y se valida que los mismos fueron implementados correctamente.

Al llegar a esta fase si no se han ido corrigiendo los errores en el transcurso del ciclo de vida del software las pruebas pueden demorar más de lo esperado y el costo del proyecto pudiera superar el presupuesto con que se dispone, ya que las mismas son una parte muy significativa del proyecto, no sólo por su importancia en el logro de resultados correctos sino por el tiempo y recursos requeridos. Se estima que demandan del orden del 60% del total del proyecto.

La preparación de las pruebas requerirá de una alta participación del personal del organismo, tanto para el diseño y preparación de datos de prueba como para su ejecución y verificación.

En el desarrollo de las pruebas deben participar representantes de distintos sectores y niveles, dado que no son tareas exclusivas del área de sistemas de la organización, sino que los usuarios son parte fundamental en la tarea de verificación de la correcta operación. La extensión de los cambios y la trascendencia que puede tener una falla requiere que en las pruebas se involucren los responsables del más alto nivel.

El hacer pruebas del software es convencionalmente clasificado en pruebas del comportamiento y pruebas estructurales: las pruebas del comportamiento enfatizan las pruebas contra los requerimientos funcionales implementados por el software, mientras que las pruebas estructurales enfatizan las pruebas basadas en la estructura interna del diseño e implementación del software.

Verificación y Validación

El Glosario Estándar de IEEE de la Terminología de Ingeniería de Software define correctitud en términos de estar libre de fallas, de satisfacer los requisitos especificados y satisfacer las necesidades

y expectativas del usuario. La correctitud del software es descrita más comúnmente como [MACCOLL 1999]:

Validación: Es la evaluación del software al final del proceso de desarrollo para asegurar el cumplimiento de las necesidades del cliente, o bien la podemos resumir en esta interrogante ¿Estamos construyendo el software correcto?

Mientras que la **verificación** es el proceso para determinar si los productos de una cierta fase del desarrollo de software cumplen o no los requisitos durante la fase anterior. La misma se resume en esta pregunta ¿Estamos construyendo correctamente el software?

Ambas, la validación y la verificación, son importantes. El fracaso de la validación constituye una ruptura en el contrato entre el que desarrolla y el cliente para cual el software se está produciendo. El fracaso en la verificación da resultado a un software que contiene faltas o fallas potenciales. Claramente, ningún fracaso es deseable.

Para la realización de los procesos de validación y verificación se utilizan técnicas que se agrupan en dos categorías: [JURISTO 2006]

Técnicas de Evaluación Estáticas: Son aquellas que buscan faltas sobre el sistema en reposo. Es decir, estudian los distintos modelos que componen el sistema software buscando posibles faltas en los mismos. Estas técnicas se pueden aplicar, tanto a requisitos como a modelos de análisis, diseño y código.

Técnicas de Evaluación Dinámicas: Generan entradas al sistema con el objetivo de detectar fallos cuando el sistema ejecuta dichas entradas. Los fallos se observan cuando se detectan incongruencias entre la salida esperada y la salida real. Las técnicas dinámicas son también conocidas como pruebas de software o *testing* y se aplican generalmente sobre código ya que es, hoy día, el único producto ejecutable del desarrollo.

Estos procesos persiguen comprobar la corrección de un sistema software a priori (sin ejecutarlo). El uso de estos “procesos” se evidencia en todo el desarrollo del producto, de una forma u otra, cuando de técnicas formales se hable. [GIL 1999] Precisamente porque estas técnicas prometen la eliminación de errores a tiempo. [POLLAN CAÑETE 2007]

La calidad siempre va a depender de los requisitos o necesidades que se desee satisfacer. Por eso, la evaluación de la calidad de un producto siempre va a implicar una comparación entre unos requisitos preestablecidos y el producto realmente desarrollado, de ahí la importancia de las técnicas de evaluación en los procesos de verificación y validación.

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

Model Checking: Inicialmente se usó en la verificación de sistemas hardware y de protocolos de comunicación. La tendencia actual es aplicar esta técnica al análisis y verificación de especificaciones de sistemas software. Este procedimiento se basa en la construcción de un modelo finito del sistema y la comprobación de que una propiedad dada se satisface en dicho modelo.

Básicamente, la demostración se produce mediante una búsqueda exhaustiva en el espacio de estados del modelo del sistema, por lo que para que termine, este espacio de estados debería ser necesariamente finito. Desde un punto de vista técnico, los trabajos de investigación del Model Checking se dirigen a la búsqueda de algoritmos y estructuras de datos más eficientes, que permitan el manejo de espacios de estados más grandes, propios de los sistemas reales. [GIL 1999]

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. [DUQUE 2000] [POLLAN CAÑETE 2007]

En la actualidad existen dos caminos para realizar Model Checking*:*

- El Model Checking temporal, desarrollado de forma independiente por Clarke y Emerson y por Queille y Sifakis en la década de los ochenta. El sistema se modela mediante un sistema de transiciones de estados finitos, mientras que la especificación se realiza a través de la lógica temporal.

Clarke y Emerson desarrollaron las herramientas **EMC** y **SMV**. Queille y Sifakis desarrollaron **CÆSAR**. Todas estas herramientas utilizaban la lógica **CTL**.

- Utilizar autómatas para modelar el sistema y para su especificación. La verificación consiste en determinar si el comportamiento del sistema es el que define su especificación. Para ello, existen varios tipos de pruebas: inclusión de lenguajes relaciones de orden y equivalencia observacional.

Las herramientas **SPIN** y **Murø** utilizan este enfoque.

El principal problema de esta técnica de verificación es la explosión de estados, es decir, la explosión del espacio de búsqueda.

Demostradores de Teoremas: Es el enfoque más antiguo en la verificación formal de sistemas. Consiste en emplear el razonamiento deductivo para extraer conclusiones a partir de axiomas o premisas cuya veracidad se conoce desde un principio o ha sido probada anteriormente.

Los demostradores de teoremas se basan en expresar tanto el sistema como sus propiedades deseadas en alguna lógica matemática. Esta lógica forma parte de un sistema formal donde se definen 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. [POLLAN CAÑETE 2007]

La principal desventaja de esta técnica 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.

A continuación se describen brevemente los demostradores de teoremas más representativos, clasificándolos según el grado de automatización que proporcionan:

- **Herramientas de deducción automáticas guiadas por el usuario:** Tienen como característica común que su operación se guía por una secuencia de lemas y definiciones, si bien cada teorema se demuestra automáticamente mediante heurísticos y la aplicación de inducción, lemas, reescritura y simplificación.

Ejemplos de este tipo de sistemas son: **Nqthm**, demostrador de teoremas de la lógica de Boyer-Moore; **ACL2**, que utiliza la lógica ACL2 (***A C****omputational* ***L****ogic for Applicative Common* ***L****isp*); Eves; **LP**; REVE y **RRL**.

- **Demostradores propiamente dichos:** Han sido usados para formalizar y verificar problemas complejos de matemáticas, así como para la verificación tanto hardware como software.

  Los ejemplos más ilustrativos de este tipo de demostradores de teoremas son:

  - ✓ **HOL:** Es un sistema demostrador de teoremas basado en lógica de orden superior, diseñado para construir especificaciones y verificaciones formales de sistemas. Ha sido aplicado en entornos académicos e industriales para el desarrollo de hardware, sistemas de tiempo real, verificación de compiladores, entre otros.
  - ✓ **Isabelle:** Es un demostrador de teoremas genérico. Posibilita la introducción de nuevas lógicas especificando su sintaxis y reglas de inferencia. Proporciona un alto grado de automatización.
  - ✓ Otros ejemplos de este tipo de demostradores son: **Coq**, **LEGO**, **LCF** y **Nuprl**.

- **Sistemas híbridos:** Su característica principal es que combinan varias técnicas de verificación. Entre los más representativos cabe citar:

  - ✓ **Analytica:** Combina la demostración de teoremas con el sistema de álgebra simbólica Matemática.
  - ✓ **PVS** y **STeP:** Combinan potentes procedimientos de decisión con Model Checking para realizar demostraciones interactivas. PVS ha sido usado con éxito para verificar diseños hardware, así como sistemas reactivos de tiempo real y tolerantes a fallos.

Por lo general es en especificación y modelación 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.

Empleo de estas técnicas

La aplicación de los métodos formales para realizar pruebas o también llamada hacer pruebas basadas en la especificación, es un tema de investigación bastante amplio e interesante,

principalmente para aquellos que producen software. Gaudel, por ejemplo, resume pruebas de programa basadas en especificaciones algebraicas, describiendo métodos para seleccionar un subconjunto finito de un conjunto de pruebas exhaustivas.

Para las notaciones basadas en modelo, Horcher y Peleska muestran como las especificaciones Z pueden ser usadas para derivar los datos de entrada de una prueba y para evaluar automáticamente los resultados de una prueba.

En la Universidad de Queensland, Austria, la Prueba Template Framework y la metodología ClassBench son técnicas bajo investigación. La Prueba Template Framework es una modelo abstracto y formal de realizar pruebas, usado para derivar una jerarquía de pruebas de información, incluyendo entradas y salidas de pruebas de una especificación formal. La Metodología ClassBench y la Prueba Template Framework proporcionan un enfoque a pruebas automatizadas de software orientado a objetos. El proyecto de prueba basado en la especificación, financiado por el Consejo Australiano de Investigación y basado en el Centro de Investigación de Verificación de Software, involucra investigadores de la Universidad Rutgers de Estados Unidos y la Universidad de Victoria, Canadá, y aspira a combinar la Prueba Template y ClassBench para crear métodos y herramientas para realizar pruebas de software orientado a objetos. [MACCOLL 1999]

En [DYKSTRA 2002] se expone cómo son aplicados los 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.

2.6 HERRAMIENTAS ORIENTADAS A LAS TÉCNICAS FORMALES

Estas herramientas son basadas propiamente en técnicas formales. Son construidas ya sea para la representación de las especificaciones como para realizar las pruebas al software. En su mayoría no son muy conocidas ya que al basarse en técnicas formales, muchas veces su funcionamiento va dirigido a estas técnicas, las cuales no son muy difundidas mundialmente por las razones planteadas en epígrafes anteriores.

2.6.1 Herramientas para la especificación de requisitos

Los métodos formales durante la etapa inicial del proceso de desarrollo de software son de gran importancia para el éxito futuro del proyecto, ya que constituyen las bases sobre las que las etapas sucesivas se apoyan. Errores producidos en esta etapa tienen un gran impacto en los costes de todo el proyecto [S.PRESSMAN 1998]. La claridad y precisión de dichos métodos es fundamental no sólo por las razones antes mencionadas sino porque, como es sabido, durante esta etapa es cuando se espera el mayor grado de interacción entre el usuario final y el desarrollador.

UML2RSL: Es una herramienta desarrollada en Java[24], lo cual la hace ser una herramienta portátil. Permite la integración de los Diagramas UML[25] con el lenguaje de especificación del método formal RAISE (RSL). Los diagramas de clase en UML que puedan ser representados en RSL tendrán una semántica formal. De esta manera, a partir de un modelo del sistema especificado por medio de un diagrama de clases en UML se puede derivar una especificación formal inicial en RSL. Esta especificación permite detectar inconsistencias en la especificación UML, además de poder usarse como base para el desarrollo de las sucesivas fases del sistema de una manera formal, controlando en todo momento su consistencia. [CHRIS 2001] [FUNES 2003]

LIRA (*Lotos Interactive Reasoning Aid*): Es un entorno transformacional que sirve de apoyo al desarrollo de sistemas distribuidos mediante el empleo de métodos formales. LIRA da soporte al desarrollo transformacional de sistemas desde la fase de captura de requisitos hasta la de implementación, eximiendo al diseñador de tareas rutinarias y permitiéndole concentrarse en los aspectos creativos del proceso. Es una herramienta que sigue el principio de refinamientos sucesivos, se parte de una arquitectura descrita en LOTOS y se refina mediante transformaciones formales verificadas.[GIL 1999]

El IFAD VDM-SL Toolbox: [LOPEZ 2004] Es una colección de herramientas para el desarrollo de especificaciones formales utilizando la última versión del estándar VDM-SL. Es usado durante las fases de especificación y diseño de un proyecto de desarrollo de software. El lenguaje completo soporta un mecanismo basado en módulos de estructuración para especificaciones grandes. Ofrece una amplia verificación de la semántica, la documentación de apoyo y análisis de depuración. Se centra en la especificación de desarrollo de sistemas industriales obteniendo así una mayor corrección.

[24] Java: Lenguaje de programación.

Ésta herramienta fue utilizada en el proyecto SPOT 4, CS-CI, en la ciudad de Francia, donde se redujo el costo significativamente [LOPEZ 2004]:

- ✓ 38 % menos de código fuente
- ✓ 36 % menos de esfuerzo total
- ✓ Generación automática de código *C* + +

2.6.2 Herramientas para el modelado del sistema

Estas herramientas son construidas con el propósito de modelar o especificar formalmente sistemas software, de manera que posteriormente brinden facilidades para comprobarlas. También contribuyen al perfeccionamiento del diseño y la arquitectura que será construida, además algunas de estas características pueden realizar verificación formal.

ZANS: Es una herramienta para la animación de especificaciones Z. Se trata de un prototipo de investigación que aún está en evolución. Las metas fundamentales de ZANS son los siguientes [DEPAUL 2002]:

- Facilita la validación de especificaciones Z.
- Perfeccionamiento del diseño y el código de síntesis sobre la base de especificaciones Z.
- Ayuda al aprendizaje del lenguaje de especificación Z.

Actualmente, tiene las siguientes características:

- Controla las especificaciones Z.
- Esquematiza la expansión de las expresiones.
- Evalúa las expresiones y predicados.
- Ejecuta los esquemas de operación.

NOTOS: Es una de las herramientas con las que se puede contar 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.

[25] UML (Lenguaje Unificado de Modelado)

AtelierB: Es una herramienta industrial que permite el uso operacional del método B para producir software de alta seguridad. Fue usado por Siemens para darle más seguridad al software METEOR. [ATELIER-B 2007], [POLLAN CAÑETE 2007]. En la ciudad de Francia con el proyecto Méteor, Matra Transport se redujo el costo para el software crítico. No se encontró ningún error durante el *testing* de 80000 líneas de código. El costo estimado para las reformas estructurales de seguridad supera ampliamente el costo que significó la verificación formal.

BToolkit: 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, entre otros.[PIECE 2002]

2.6.3 Herramientas para construcción del sistema

Las mismas ayudan a garantizar la eficiencia del software, ya que contribuyen a la construcción de un software libre de errores, robusto y con un rendimiento previsible. Ejemplo de ello es la siguiente herramienta:

La herramienta **ArgoSPE** contribuye a la construcción de un software con rendimiento predecible especificando y analizando la conducta cuantitativa desde las fases de inicio de desarrollo del software. Ésta ha sido implementada como un conjunto de módulos Java, que son *"plugged"* en la herramienta de código abierto ArgoUML. Esta herramienta será tratada más adelante. [SANTIAGO]

2.6.4 Herramientas para la verificación y validación de requisitos

Al validar y verificar los requisitos estamos garantizando corrección en el software. Una herramienta de verificación ayuda a comprobar que el sistema especificado sea el correcto.

LTSA (Analizador Etiquetado de Transición): Es un producto de IBM de licencia libre, es una versión para Eclipse. Esta es una herramienta de verificación para sistemas concurrentes. Inspecciona automáticamente que la especificación de un sistema concurrente satisface las propiedades requeridas de su comportamiento. Además LTSA soporta animación de especificación para facilitar exploración interactiva del comportamiento del sistema.[LONDON 2007]

Un sistema en LTSA es modelado como un grupo de interacciones de máquinas de estados finitos. Las propiedades requeridas del sistema son también modeladas como máquinas de estados. LTSA realiza análisis de asequibilidad del componente para exhaustivamente ir en busca de violaciones de las propiedades deseadas. Cada componente de una especificación es descrito como un Sistema Designado de Transición (LTS), lo cual contiene a todos los estados que un componente puede alcanzar y las demás transiciones que puede arribar. Sin embargo la descripción explícita de un LTS en términos de sus estados, etiquetas de acción y relación de transición es difícil para sistemas grandes. Consecuentemente, LTSA soporta una notación de álgebra de proceso (FSP) para la descripción de la secuencia de comportamiento del componente. La herramienta permite al LTS correspondiente a una especificación FSP ser mirada gráficamente. [LONDON 2007]

La etapa de la especificación de requisitos es la más importante para que un sistema pueda responder correctamente a las solicitudes realizadas por el cliente. Para que un software pueda hacer lo que desea el usuario, en otras palabras para que opere correctamente, es necesario entender qué es lo que realmente espera el cliente del producto, he aquí la importancia de la validación de requisitos para garantizar la corrección del software.

Model Checker: 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] [POLLAN CAÑETE 2007]

Hoy por hoy existen múltiples herramientas de Model Checker, tanto comerciales como de dominio público. En el ámbito académico, se destaca principalmente Spin y sus variantes:

Spin1: Es una herramienta de código abierto que se emplea en la verificación de errores de diseño en sistemas distribuidos. Utiliza enumeración explícita de estados y reducción parcial de orden. 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.[POLLAN CAÑETE 2007]

JKingQA: Es una herramienta de análisis estático, 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, a la eficiencia y corrección del software. [POLLAN CAÑETE 2007]

C Global Surveyor (CGS): 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.

2.6.5 Herramientas para el entorno de pruebas

Las Técnicas de Descripción Formal (TDF) se han utilizado para especificar protocolos de seguridad y evaluar la vulnerabilidad de dichos protocolos frente a distintos ataques. Ejemplos de este tipo de análisis formal es la siguiente herramienta:

El Analizador de ataques (A.N.A.): Es una herramienta de análisis de protocolos de seguridad basada en la especificación formal de protocolos y en su interpretación automática como reglas de Prolog. Tiene diseñada una especificación formal lo suficientemente completa y concisa. Permite definir la máquina de estados que corresponde a un protocolo criptográfico, así como incorporar de forma flexible mecanismos y funciones de seguridad (procedimientos de generación de números aleatorios, sellos de tiempo, cifrado simétrico, asimétrico entre otros).Teniendo en cuenta lo anteriormente planteado con esta herramienta se garantiza la integridad del software. Se analizará con más detalle en el próximo capítulo (Ver anexo 6 y 7).[MENGUAL]

JTest, C++Test y Insure++ de Parasoft : Herramientas de Parasoft que ayudan a prevenir errores por medio de su capacidad de análisis estático personalizable, que permite hacer cumplir automáticamente alrededor de 300 estándares de codificación de la industria, crear y hacer cumplir estándares propios de codificación, o construir estándares destinados a un proyecto o grupo en particular. [ALS 2007]

QACenter Performance Edition de Compuware: Ayuda a realizar pruebas realistas, en profundidad y reproducibles para aplicaciones de *e-business*, ERP y cliente/servidor. Gracias a una potente combinación de herramientas para pruebas de carga, gestión de datos y monitorización de servidores,

QACenter Performance Edition permite a las empresas buscar y resolver rápidamente problemas de rendimiento., lo que contribuye a garantizar la corrección del software. [ALS 2007]

Security Tester de Fortify: Proporciona pruebas de seguridad eficaces a los equipos de desarrollo y calidad, permitiéndoles verificar la adecuación a los estándares de seguridad y posibles vulnerabilidades en el código de sus aplicaciones antes de su despliegue, garantizando de este modo la integridad del software. [ALS 2007]

Fortify Software Security Manager: Es una herramienta de control de seguridad para la gestión de riesgos derivados del software, está orientada para los equipos de desarrollo y seguridad. Además de proporcionar informes flexibles, esta herramienta aporta una gestión centralizada de reglas, políticas de seguridad y alertas, garantizando la integridad del software. [ALS 2007]

TrackRecord de Compuware: Se ajusta a cualquier proceso de desarrollo y pruebas, ofreciendo un sistema de rastreo que ayuda en la identificación y resolución de defectos software, garantizando la corrección e integridad del software. [ALS 2007]

2.7 CONCLUSIONES DEL CAPÍTULO

En este capítulo se ha realizado un estudio de los indicadores de calidad que se deben garantizar en todo el proceso de desarrollo de software para asegurar los factores externos de calidad: corrección, eficiencia, e integridad, evidenciando la importancia que tienen estos factores para la obtención de un software con óptima calidad.

Aunque no han sido muy empleadas en el desarrollo de software, las técnicas formales han alcanzado excelentes resultados en los proyectos que las han empleado. Un camino para la obtención de un software correcto, eficiente e íntegro debe ser la vinculación de estas técnicas basadas en el uso de las matemáticas con las técnicas tradicionales.

El uso de herramientas que emplean las técnicas formales es importante para obtener una mayor calidad en el producto software, evitar errores que comúnmente cometen los humanos en el empleo de estas técnicas, reducir el tiempo estimado de desarrollo, obtener un mayor beneficio del producto y disminuir el coste de producción.

CAPÍTULO III: PROPUESTA DE HERRAMIENTAS Y TÉCNICAS FORMALES PARA AUMENTAR LA CORRECCIÓN, EFICIENCIA E INTEGRIDAD DEL SOFTWARE EN LA UCI

3.1 INTRODUCCIÓN

La calidad del software es una de las problemáticas más importantes en los procesos de desarrollo de software. Garantizar el correcto funcionamiento bajo situaciones no determinadas es una tarea que tiene que ser realizada con cuidado extremo.

Muchas veces se produce software que es satisfactorio, ya que resuelve el problema que el cliente plantea, sin embargo, no siempre el producto tiene buena calidad, y esto se debe a que precisamente cuando son ejecutados, o llevan un período de tiempo en explotación afloran las fallas y el costo de su mantenimiento es elevado.

3.2 TÉCNICAS EMPLEADAS EN LA UCI

En la Universidad de las Ciencias Informáticas se cuenta de forma general con poca experiencia en la producción de software, existe un gran número de personal involucrado en el proceso de construcción del producto que han logrado resultados satisfactorios. Sin embargo según la encuesta realizada (ver anexo 5) no se utilizan las técnicas formales. En cada proyecto productivo se adoptan las técnicas tradicionales según les sea más conveniente. Ejemplo de ellas son:

- **Entrevista:** Es una técnica muy aceptada dentro de la Ingeniería de Requisitos y su uso está ampliamente extendido. Las entrevistas le permiten al analista tomar conocimiento del problema y comprender los objetivos de la solución buscada. A través de esta técnica el equipo de trabajo se acerca al problema de una forma natural. Básicamente, la estructura de la entrevista abarca tres pasos: identificación de los entrevistados, preparación de la entrevista, realización de la entrevista y documentación de los resultados (protocolo de la entrevista). Las entrevistas, sin embargo, no son una técnica sencilla de aplicar. Requieren que el entrevistador sea experimentado y tenga capacidad para elegir bien a los entrevistados y obtener de ellos toda la información posible en un período de tiempo siempre limitado. Bajo este aspecto la preparación de las entrevistas representa un papel esencial.

- **Brainstorming** (Tormenta de ideas): En la UCI se utiliza en la captura de requisitos, esta es una técnica de reuniones en grupo cuyo objetivo es que los participantes muestren sus ideas de forma libre. Consiste en la mera acumulación de ideas y/o información sin evaluar las mismas. Además suele ofrecer una visión general de las necesidades del sistema, pero normalmente no sirve para obtener detalles concretos del sistema, por lo que suele aplicarse en los primeros encuentros.
- **Casos de Uso:** Esta técnica es la más empleada en el desarrollo de software en la universidad. Se utiliza como técnica para la captura de requisitos. Permite ver el alcance de los requisitos funcionales expresados como casos de uso. Un caso de uso describe la secuencia de interacciones que se producen entre el sistema y los actores del mismo para realizar una determinada función. La ventaja esencial de los casos de uso es que resultan muy fáciles de entender para el usuario o cliente, sin embargo carecen de la precisión necesaria si no se acompañan con una información textual o detallada con otra técnica como pueden ser diagramas de actividades.
- **Cuestionarios y Check Lists (listas de chequeo):** Es otra de las técnicas utilizadas en la universidad. Esta técnica requiere que el analista conozca el ámbito del problema en el que está trabajando. Consiste en redactar un documento con preguntas cuyas respuestas sean cortas y concretas, o incluso cerradas por unas cuantas opciones en el propio cuestionario (Check List). Este cuestionario será cumplimentado por el grupo de personas entrevistadas o simplemente para recoger información en forma independiente de una entrevista.

Existen otras técnicas para la captura de requisitos, incluso también es común encontrar alternativas que combinen muchas de estas técnicas. Sin embargo, las presentadas ofrecen un conjunto representativo de las más utilizadas de manera no formal para la captura, definición y validación de requisitos de los productos de software en la UCI.

3.3 RESULTADOS DE LA ENCUESTA

En el mes de mayo del año 2008 se realizó una encuesta a un total de 40 estudiantes y profesores de las 10 facultades de la UCI vinculados a los proyectos productivos. Con el objetivo de conocer qué métodos de emplean en el proceso de producción de software en la UCI y así poder contribuir a un mejoramiento de la calidad del producto. Una vez analizadas las encuestas se obtuvieron los siguientes resultados:

Solo el 40% del personal que laboran en los proyectos tienen experiencia en la producción de software. Un 20% valora de buena la calidad del software desarrollado en la UCI, un 75% de regular y un 5% la valora de poca calidad. Solamente el 10% de los proyectos encuestados destacaron que se realizan revisiones de calidad a sus proyectos varias veces, un 72% opinan que son pocos los controles de calidad y un 18% no saben acerca de los controles de calidad que se han realizado a su proyecto. Se desconoce completamente de las técnicas formales para el desarrollo de software, en todos los casos se utilizan técnicas tradicionales. Un 53% de los encuestados reflejaron que en su proyecto las pruebas se realizan al finalizar el proyecto, un 37% al final de cada etapa y un 10% por etapas. (Ver anexos 6, 7,8 y 9)

3.4 PROPUESTA DE TÉCNICAS FORMALES

Las técnicas formales son uno de los métodos existentes en la Ingeniería del Software para ayudar a la creación de sistemas de elevada complejidad, sin embargo, a pesar de su dificultad se pretende con las mismas alcanzar los parámetros de eficiencia y eficacia deseados. El uso de métodos formales no garantiza, a priori, la corrección del software, pero es una buena práctica que permite alcanzar mejores resultados en la construcción de sistemas complejos, ya que permite revelar inconsistencias y ambigüedades.

En el pasado el uso de estas técnicas no estaba generalizado debido a los problemas que llevaban asociados, como la complejidad de los métodos, su baja capacidad para escalar, entre otros. Actualmente, estas técnicas están aumentando su grado de aceptación y mejorando su capacidad de éxito lo que convierte a los métodos formales en una técnica que no debe dejarse de tener en cuenta para la creación de sistemas computacionales de calidad.

De las entrevistas realizadas a líderes de proyectos, estudiantes, profesores y al personal involucrado en la producción de software en la UCI se pudo conocer que el alto índice del fracaso en el desarrollo de software tiene origen o son causados por actividades relacionadas con los requisitos, muchas veces esto se debe a la falta de participación de los usuarios, a requisitos incompletos y frecuentemente a cambios en los requisitos iniciales. Además la corrección de requisitos defectuosos implica mucho esfuerzo sobre otros productos del proyecto.

Por lo expresado anteriormente, se propone para la validación de requisitos la técnica **Model Checking**. La misma permite demostrar matemáticamente, a través de la verificación, que un sistema

cumple con las propiedades deseadas, que los refinamientos de la especificación inicial son correctos, o que la implementación final alcanza los objetivos propuestos. Es posible automatizar totalmente sus 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. La misma también tiene influencia en los factores externos estudiados en el presente trabajo por lo que se propone el uso de la misma. Esta técnica fue propuesta en [POLLAN CAÑETE 2007] para el factor fiabilidad.

Demostradores de teoremas: La técnica que emplean los demostradores de teoremas se basa en expresar tanto el sistema como sus propiedades deseadas en alguna lógica matemática. Una propiedad se define como un teorema, cuya veracidad es el objetivo a demostrar. Esta técnica en conjunto a Model Checking son las más difundidas en la verificación formal pero a diferencia del Model Checking los demostradores de teoremas pueden manejar espacios de estados infinitos. Permite una mejor comprensión de los requisitos del sistema, reduciendo errores y omisiones en la definición de los mismos. Además proporciona la base para un diseño elegante. Esta técnica es una de las más usadas, ya que la forma de trabajo es muy ordenada. Esta técnica también fue propuesta en [POLLAN CAÑETE 2007] para el factor fiabilidad.

SDL: Es un lenguaje amplio y desarrollado, tanto en su utilidad (desde la fase de especificación de requisitos hasta la de implementación) como en su ámbito de aplicación (sistemas de telecomunicación, control ferroviario, aplicaciones médicas, entre otras).

En SDL, los sistemas se representan estructuralmente mediante una jerarquía de bloques y procesos que intercambian señales entre ellos y con el entorno. Se trata de un modelo orientado a objetos que permite la especificación de subsistemas dentro de los bloques, lo cual favorece el desarrollo por refinamientos sucesivos. La descripción del comportamiento de los sistemas está basada en máquinas de estados finitos extendidas (que representan los procesos) que se comunican entre sí y con el entorno. Ello lo convierte en un lenguaje muy apropiado para representar sistemas basados en estímulo-respuesta. SDL es, probablemente, la técnica formal más empleada en la industria.

3.4.1 Propuesta para la Modelación del sistema

LOTOS: Es una de las técnicas constructivas basada en el álgebra de procesos especialmente acoplada para la fase de diseño del sistema. Permite realizar a través de un análisis técnico la verificación, validación y desarrollo del mismo. Esta técnica es especialmente útil en la especificación de sistemas distribuidos y concurrentes, como son los protocolos y servicios de telecomunicaciones. Se utiliza para verificar la seguridad de protocolos y servicios de telecomunicación. Como técnica formal que es, la semántica formal de LOTOS permite realizar descripciones completas, claras, consistentes y no ambiguas del sistema a especificar.

La fuerte base matemática que sustenta la semántica formal de LOTOS hace posible el desarrollo y aplicación de métodos de validación y verificación de propiedades y equivalencias. Ello es de vital importancia para lograr el objetivo prioritario de obtener sistemas altamente fiables, íntegros y eficientes. A través de estos mecanismos, no sólo se puede garantizar que la especificación cumple una serie de requisitos o propiedades de interés, sino que se puede llegar a determinar la corrección o conformidad de una implementación respecto a una especificación. En cualquier caso, el empleo de esta clase de métodos ayuda a descubrir, más rápidamente, la existencia de posibles errores o disfuncionalidades en la especificación del sistema.

VDM: El Vienna Development Method es uno de los métodos más experimentados para la especificación formal y el subsiguiente desarrollo de aspectos funcionales de soportes lógicos. El elemento central es su lenguaje de especificación VDM-SL. Está siendo estandarizado por la ISO (Institución Internacional de Estándares) y BSI (Institución Británica de Estándares). Este método permite desde la etapa de requisitos evitar que se propaguen errores a etapas posteriores, analiza incoherencias en el análisis y diseño de sistemas ya que utiliza una notación matemática precisa y rigurosa. Facilita la implementación y la generación automática de código, así como la verificación y validación de requisitos, lo cual garantiza una mayor corrección al software.

3.5 PROPUESTA DE HERRAMIENTAS BASADAS EN TÉCNICAS FORMALES

3.5.1 Propuesta para la eficiencia del software

ArgoSPE es una herramienta para la evaluación de rendimiento de sistemas de software, con licencia GNU (*General Public License*). Ésta ha sido implementada como un conjunto de módulos Java, que son *"plugged"* en la herramienta de código abierto ArgoUML.

La especificación de rendimiento del software con un lenguaje semiformal como UML, requiere su integración con un formalismo de modelación de rendimiento, como las Redes de Petri Estocásticas Generalizadas (GSPN), en inglés *Generalize Stochastic Petri Nets*. Sobre esta base la herramienta CASE ArgoSPE traslada los diagramas UML a GSPN, y a partir de ello, evalúa el rendimiento de los sistemas.

Uno de los objetivos del rendimiento es el grado en que un sistema de software satisface sus objetivos de consumo de recursos y de tiempo, este último aspecto, se vuelve crítico en algunas aplicaciones de tiempo real. Otros de los indicadores de rendimiento según el modelo de calidad FURPS son: la eficacia y la velocidad del procesamiento, estos elementos están intrínsecamente vinculados a la eficiencia del software, factor de calidad tratado en esta tesis e imprescindible para que un sistema tenga calidad.

UML es el lenguaje más utilizado para la modelación de sistemas a nivel mundial y también el más empleado en los proyectos productivos de la UCI, por lo que se propone la herramienta libre ArgoSPE de licencia GNU, que es un *plug-in* que se integra a la herramienta de modelado ArgoUML bajo licencia BSD. Esta herramienta contribuye a que los desarrolladores del sistema que tradicionalmente han utilizado los métodos semiformales de modelado se sientan más cómodos con la integración de ésta a técnicas de especificación formal automatizadas. Tiene un ambiente de desarrollo interactivo, amigable, se encuentra en varios sitios de descarga libre[26], colaborando con la migración a software libre, objetivo propuesto por la Universidad de las Ciencias Informáticas. Tiene un manual de usuario lo cual facilita el entendimiento de esta herramienta, además posee una guía de instalación donde uno de

[26] http://argospe.tigris.org

los requisitos que posee es tener instalada una máquina virtual de Java. Fue instalada en el Polo de Teleformación y probado su funcionamiento (ver anexo 10).

3.5.2 Propuesta para la integridad del software

Los modelos formales de seguridad han evolucionado en paralelo con el desarrollo de los sistemas informáticos (software, hardware, sistemas operativos), así como con la tecnología y extensión de las redes de datos. Tradicionalmente, los modelos formales han tratado el problema del control de acceso en sistemas individuales. Sin embargo, el notable impulso de las redes locales y su interconexión con redes remotas, ha dado lugar a nuevas y más sofisticadas amenazas asociadas a la distribución de la información (grabaciones, retransmisiones, suplantaciones, entre otras). Todo ello ha conducido al desarrollo de nuevos mecanismos y funciones de seguridad para proteger la integridad de la información en tránsito en los sistemas abiertos, y no solo contemplar los problemas de control de acceso a recursos en los sistemas individuales.

Teniendo en cuenta los resultados alcanzados en el transcurso de la investigación se propone la herramienta de análisis de protocolos de seguridad denominada **Analizador de ataques (*A*.N.A.)**, para garantizar la integridad de los datos de los productos software desarrollados en la UCI en la fase de Prueba. Esta herramienta está basada en la especificación formal de protocolos y en su interpretación automática como reglas de Prolog.

El objetivo fundamental de A.N.A consiste en estudiar de forma automática la vulnerabilidad del protocolo explorando todos los comportamientos posibles del mismo. Todo ello de acuerdo con la especificación formal del protocolo y la información potencialmente grabada por un intruso.(Ver anexo 11 y 12).

Existen protocolos que anteriormente sus vulnerabilidades habían sido probadas de forma teórica, ahora estas pueden ser probadas y analizadas de manera automática a partir de su traducción a reglas, ejemplo de ello son los protocolos SSH o AKA. El sistema se ha concebido como un entorno de pruebas que sirva de base para el análisis y evaluación de ataques reales a un protocolo de seguridad.

3.5.3 Propuesta para la corrección del software

NOTOS es un generador de prototipos para especificaciones LOTOS. El generador toma como entrada una especificación de un sistema escrita en LOTOS y como salida obtiene un prototipo ejecutable escrito en lenguaje C con el que simulamos el comportamiento del sistema. De esta forma, se puede verificar si la especificación del sistema que se realizó en LOTOS es adecuada, garantizando con esta herramienta el factor externo de calidad: corrección, ya que la misma satisface los requisitos del usuario. Además realiza un chequeo léxico, sintáctico y semántico generando un prototipo. Esto representa una gran ayuda, ya que una vez realizada la labor de especificación de un sistema cualquiera, se puede verificar el comportamiento mediante una simulación con el prototipo.

3.6 CONCLUSIONES

La integración de los métodos formales con los métodos tradicionales para garantizar la corrección, eficiencia e integridad del software desarrollado en la UCI, contribuirá a elevar la preparación del personal y facilitará el desarrollo de software con una mejor calidad.

Existe un desconocimiento por parte de los desarrolladores de software de la Universidad de las Ciencias Informáticas sobre las técnicas formales, esto se debe en gran medida a que no forman parte del plan de estudio; no existe personal preparado capaz de impartir estos conocimientos y además se desconocen las ventajas que estos métodos proporcionan tanto a clientes como a desarrolladores.

CONCLUSIONES GENERALES

En este trabajo se analizaron aspectos relacionados con el creciente avance de la industria de software y la necesidad de construir software con calidad empleando herramientas basadas en técnicas formales, vinculadas a factores externos de calidad. Se cumplieron los objetivos del trabajo, permitiendo arribar a las siguientes conclusiones:

- Se hace una propuesta de técnicas formales y herramientas basadas en ellas que inciden en la corrección, la eficiencia e integridad en la producción de software.
- No existe una técnica formal que se ajuste a todas las fases del proceso de desarrollo de software.
- Las herramientas que integran el lenguaje semiformal UML con un formalismo facilitan el tránsito hacia el empleo de técnicas formales.
- En el estudio realizado, se evidenció que el uso de las técnicas formales desde fases tempranas del desarrollo de software favorece en gran medida, la corrección, eficiencia e integridad del software.
- Las técnicas formales son métodos muy útiles para eliminar ambigüedades, validar y verificar que un sistema satisface sus requerimientos.

RECOMENDACIONES

Para dar continuidad al trabajo, se recomienda:

- El estudio de los métodos formales y herramientas orientadas a los factores de calidad restantes para que sean incluidas en la propuesta actual.
- La profundización en el estudio de las técnicas, lenguajes de especificación formal y las herramientas que las sustentan y que se incluyan dentro del plan de estudio de la universidad y en la superación de pregrado y postgrado.
- Hacer extensiva la propuesta de este trabajo para las empresas cubanas productoras de software y las universidades.
- Comenzar a emplear de manera inmediata las herramientas y las técnicas propuestas, en los proyectos de desarrollo de software de la UCI.
- Desarrollar un sistema automatizado que integre las técnicas formales propuestas.
- Utilizar el sistema propuesto en el punto anterior en proyectos productivos de la UCI como forma de validación de la propuesta.

REFERENCIAS BIBLIOGRÁFICAS

8402, I. *Calidad en Ingenieria de Software*, 2002.

ALS. *Herramientas para el entorno de pruebas* 2007. [Disponible en: http://www.als-es.com/home.php?location=herramientas/entorno-pruebas

ARTUR BORONAT, J. I., JOSÉ Á. CARSÍ, ISIDRO RAMOS, ABEL GÓMEZ. *Del método formal a la aplicación industrial en Gestión de Modelos: Maude aplicado a Eclipse Modeling Framework1*, 2003. 6.

ATELIER-B "Atelier B", 2007.

AVENDAÑO, D. *Métodos formales en el proceso de desarrollo de software*, 2006. [2008]. Disponible en: http://impreso.elnuevodiario.com.ni/2006/12/24/informatica/64098

B. *"Tools to be presented".En sitio: B2007. The 7th International B Conference.*, 2007. [Disponible en: http://lifc.univ-fcomte.fr/b2007/pages/tools.htm

BJORNER, D. *"32 Years of VDM from Earliest Days via Adolescence to Maturity" Universidad Técnica de Dinamarca.*, 2006.

BOTON, M. *Introducción a los métodos formales*, 2008. [1]. Disponible en: "http://campusvirtual.unex.es/cala/epistemowikia/index.php?title=Introducción_a_los_Métodos_Formales"

CABRERA, E. V. *Software reliability methods*. Madrid, Universidad Polit¶ecnica de Madrid, 2006. p.

CALIDAD, L. N. D. *Calidad de software.Herramientas*, 2008. [Disponible en:

CAÑETE, P., SUSEL; CHANG, AROCHA, YAIMA. *Técnicas formales para el análisis y predicción del comportamiento fiable del software.* Cuba, Universidad de las Ciencias Informáticas, 2007. p.

CARRASCO, O. M. F. *Informes Técnicos*, 1995. [Disponible en: http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm

CERNUDA, D. R., AGUSTÍN; LABRA, GAYO, JOSE, EMILIO; CUEVA, LOVELLE, JUAN, and MANUEL. "Verificación y validación mediante un modelo de componentes". , 2001: (pp. 1-3 y 5).

CERVERA PAZ, A. N. M., 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*, 1997.

CMMIv1.2. *CMMI para el desarrollo .Mejora de procesos para mejores productos*, 2006.

COUTIN, N. L. N. C. R. *"Diccionario de Informática"*. Editorial Científico -Técnica., 2005. p.

CUEVA, J. M. *Calidad del Software.Departamento de Informática Universidad de Oviedo España.*, 1999.

CHRIS, G. RAISE Tool User Guide, 2001: 135.

DEPAUL, U. Z Animations, 2002.

DUQUE, J. G. *Especificación, verificación y mantenimiento de requisitos funcionales con técnicas de descripción formal.* España, Universidad de Vigo

Dpto. de Tecnologías de las Comunicaciones ETSI de Telecomunicación, 2000. p.

DYKSTRA, J. "Verificación y Validación de Software con Destiny: Un enfoque paralelo a la prueba automática de teoremas", 2002.

ESCALONA, M. J. K., NORA. *"Ingeniería de Requisitos en Aplicaciones para la Web. Un estudio comparativo" Departamento de Lenguajes y Sistemas Informáticos en Escuela*. España, Universidad de Sevilla, 2002. p.

FERNÁNDEZ CARRASCO, O. M., GARCÍA LEÓN, DELBA Y BELTRÁN BENAVIDES, ALFA. *Un enfoque actual sobre la calidad del software.*, 1995. p.

FRANCISCO J. , S. D., DIEGO J. BODAS SAGI, VALENTÍN POZO LLORENTE. *"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.*, 2005.

FRANCISCO JOSÉ GALÁN MORILLO, J. M. C. V. *Métodos Formales Orientados a Objetos*: 20.

FUNES, A. D., ARÍSTIDES Integración de Modelos en UML y Especificaciones Formales:Transformaciones de OCL a RSL, 2003.

GALLARDO, M., MARÍA, DEL MAR; MERINO, GÓMEZ,PEDRO "Métodos para la Construcción de Software Fiable: Interpretación Abstracta", 2006.

GALLUD, J. A. G. C., J. M. "NOTOS: Un Generador de Prototipos para Especificaciones LOTOS" DISCA, 2007.

GIL, A. S. *Diseño y verificación de sistemas distribuidos mediante la aplicación combinada de métodos formales.* . España, Universidad de Vigo, 1999. p.

GOGUEN, J. *"The OBJ Family".Sitio del Departamento de Ciencia de la Computación e Ingeniería. Universidad de California. San Diego, USA.*, 2005.

GÓMEZ, L., PRISCILA,RUTH, HUERTA,DEL JUNCAL ,JORGE LUIS *Herramientas Case*, 2008.

GONZALES URMACHEA, M. *"Redes de Petri"* ? [2008]. Disponible en: <http://www.monografias.com/trabajos14/redesdepetri/redesdepetri.shtml#mo>

GONZALEZ, C. *Conceptos generales de calidad total*, 2007. [Disponible en: http://www.monografias.com/trabajos11/conge/conge.shtml

GORM LARSEN, P. *"The VDM-SL Examples Repository"*, 2006.

GRENOBLE "Outils industriels / Industrial Tools", 2007.

I-SOL. *Cómo obtener un Software con Calidad*, 1999. [Disponible en: http://www.i-sol.com.ar/pg005.html

IEEE, S. C., Ed. *"IEEE Standard for Software Quality Assurance Plans"*, 1998.

INDUSTRIAL, I. U. T. and "Calidad Total-Reingeniería-Normas ISO 9000." 1999.

Infocalidad. 2004. [Disponible en: http://www.infocalidad.net/gest_calidad_def/definicion.asp

INSTITUTO, W. "Norma de Calidad. Gestión de La Calidad o Excelencia", 2006.

ISO/IEC. *Technology - Software product quality*, 2005.

J. MCCALL, P. R., G.WALTERS. *Factors in Software Quality*, 1977. 3 vols.

JURISTO, N. M., ANA M.; VEGAS, SIRA "Técnicas de evaluación del software", 2006.

LARCH. *"Larch Home Page"*, 2001.

LEAVENS, G. T. *"What good is it?" Departamento de Ciencias de la Computación de la Universidad del estado de Iowa.*, 1999.

LONDON, I. C. *LTSA Eclipse and WS-Engineer LTSA Eclipse y WS-Ingeniero* 2007. [2008]. Disponible en:http://translate.google.com.cu/translate?hl=es&sl=en&u=http://eclipse-plugins.info/eclipse/plugin_details.jsp%3Fid%3D1243&sa=X&oi=translate&resnum=1&ct=result&prev=/search%3Fq%3Dtool%2BLTSA%2Blicense%26hl%3Des%26sa%3DG o http://www.doc.ic.ac.uk/ltsa/eclipse/

LOPEZ, P., CARLOS, GUSTAVO Métodos formales en la ingeniería de software:en busca de la bala de plata, 2004.

MACCOLL, I. C., DAVID. *"Correctitud de la Interfaz con el Usuario."Sitio Crossroads.*: *The ACM Student Magazine*, 1999.

MCDONALD LANDAZURI, B. A. *¨Definición de perfiles en Herramientas de Gestión de Requisitos¨*. Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del Software. Madrid, España, Facultad de Informática de la Universidad Politécnica de Madrid, 2005. p.

MEDINA PATÓN, D. E. F. *"Mantenimiento del Software" Tipos de Mantenimiento y Coste Relativo. 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.*, 2007.

MENGUAL, L., BARCIA, N., FERNÁNDEZ, C., JIMÉNEZ, E., SETIÉN, J., YAGÜEZ, J. *El analizador de ataques (A.N.A.): Herramienta de análisis de la vulnerabilidad de protocolos de seguridad mediante su traducción a un sistema basado en reglas.* .

MESTRAS, J. P. Garantía de calidad del software, 2004.

MUÑOZ, C. *Métodos formales, por ejemplo.Computer Science Laboratory*, 2008.

NAPAL, I. B. E. I. *"Las pruebas de software, su aplicación al Config. Case".* . Ciudad de la Habana, Instituto Superior Politécnico "José Antonio Echeverría". 2003. p.

NASA. *"La computadora lo puede verificar"*, 2003. [Disponible en: http://www.nasa.gov/centers/ames/spanish/research/exploringtheuniverse/exploringtheuniverse computercheck.html

PALACIO, J. *"Gestión y modelos para la eficiencia en empresas de desarrollo software". Sitio BAQUIA: Centro de Conocimiento.*, 2005. [Disponible en: http://www.baquia.com/noticias.php?id=9651

PAREDES, A. P. R. *Construyendo software de alta calidad*, 2005. [Disponible en: http://www.elguille.info/colabora/NET2005/Percynet_ConstruyendoSoftCalidad

PAZOS ARIAS, J. J. *"Especificación Formal. Técnicas para la especificación no ambigua del software"*. Ingeniería del software de comunicaciones., 2000. p.

PIECE, H., KING. *"The B-Toolkit"*, 2002. [Disponible en: http://www.b-core.com/btoolkit.html

POLLAN CAÑETE, S. C. A., YAIMA. *Técnicas formales para el análisis y predicción del comportamiento fiable del software.* Cuba, Universidad de las Ciencias Informáticas, 2007. p.

QUESADA., M. R. M. C. J. P. F. R. M. O. C. Z. C. T. C. V. C. J. G. R. *Experiencias de la Aplicación de la Ingeniería de Software en Sistema de Gestión.*: *Revista Cubana de Informática.*, 2001.

ROMERO, L., JENNY;SALCEDO, RODRÍGUEZ,YORDANKY. *Propuesta de Modelo de Calidad para Portales Web.* Ciudad Habana, Universidad de las Ciencias informáticas., 2007. 107. p.

S.PRESSMAN, R. *Ingenieria de Software.Un enfoque práctico.*, 1998. p.

SÁNCHEZ, J. B., 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"*, 2002.

SÁNCHEZ PALMA, P. *"Animación de Especificaciones OASIS mediante redes de Petri Orientadas a Objeto" Tesis Doctoral de Informática.Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, España.* , 2002. [2008]. Disponible en: http://www.dsic.upv.es/users/oom/docs/tesisppalma.pdf

SANTIAGO, P., MARIO, DISTEFANO, ANTONIO, PASEO, ATILIO, RANZUGLIA UML y REDES DE PETRI EN LA EVALUACION DE PERFORMANCE DE SISTEMAS.

SOMMERVILLE, I. "Verificación y Validación" (Prentice Hall). 2000, 6 ta edición

TENNEY, R. L. *"Estelle Information"*, 2000.

UNIVERSIDAD, R., JUAN, CARLOS Calidad del software.Ingeniería del Software I, 1990.

VIZCAÍNO, A. G., FELIX ÓSCAR; CABALLERO, ISMAEL "Una Herramienta CASE para ADOO: Visual Paradigm. Análisis y Diseño Orientado a Objetos" Prácticas Ingeniería delSoftware 3º, 2006.

BIBLIOGRAFÍA CONSULTADA

8402, I. *Calidad en Ingeniería de Software*, 2002. [Disponible en: http://dmi.uib.es/~bbuades/calidad/sld001.htm

A., H. A. L. *Notas sobre el uso de Métodos formales en los procesos de desarrollo de software?* : 6.

ALSES. *Alses*? [2008]. Disponible en: http://www.alses.com/index.php

ARTUR BORONAT, J. I., JOSÉ Á. CARSÍ, ISIDRO RAMOS, ABEL GÓMEZ. *Del método formal a la aplicación industrial en Gestión de Modelos: Maude aplicado a Eclipse Modeling Framework1*, 2003. 6.

ATELIER, B. *Atelier de génie logiciel permettant de développer des logiciels prouvés sans défaut?* [2008]. Disponible en: http://www.atelierb.eu/

AVENDAÑO, D. *Métodos formales en el proceso de desarrollo de software*, 2006. [2008]. Disponible en: http://impreso.elnuevodiario.com.ni/2006/12/24/informatica/64098

B. *"Tools to be presented"*, The 7th International B Conference" Publicado: 2007. , 2007. [2008]. Disponible en: < http://lifc.univfcomte.fr/b2007/pages/tools.htm>

BJORNER, D. *"32 Years of VDM from Earliest Days via Adolescence to Maturity" Universidad Técnica de Dinamarca.*, 2006.

BOTON, M. *Introducción a los Métodos Formales*, 2008. [1]. Disponible en: "http://campusvirtual.unex.es/cala/epistemowikia/index.php?title=Introducci%C3%B3n_a_los_M%C3%A9todos_Formales"

B-TOOLKIT. *B-Toolkit*, 2002. [2008]. Disponible en: http://www.b-core.com/btoolkit.html

BOWEN, J. *"Formal Specification and Documentation using Z: A Case Study Approach."* 1996.

CABRERA, E. V. *Software reliability methods*. Madrid, Universidad Politécnica de Madrid, 2006. p.

CALIDAD, L. N. D. *Calidad de software. Herramientas*, 2008. [Disponible en:

CARRASCO, O. M. F. *Informes Técnicos*, 1995. [Disponible en: http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm

CERNUDA, D. R., AGUSTÍN; LABRA, GAYO, JOSE, EMILIO; CUEVA, LOVELLE, JUAN, and MANUEL. "Verificación y validación mediante un modelo de componentes". , 2001: (pp. 1-3 y 5).

CERVERA PAZ, A. N. M., 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*, 1997.

CHRIS, G. RAISE Tool User Guide, 2001: 135.

CMMIV1.2. *CMMI para el desarrollo .Mejora de procesos para mejores productos*, 2006.

COUTIN, N. L. N. C. R. *"Diccionario de Informática"*. Editorial Científico -Técnica., 2005. p.

CUEVA, J. M. *Calidad del Software. Departamento de Informática Universidad de Oviedo España.*, 1999.

DUQUE, J. G. *Especificación, verificación y mantenimiento de requisitos funcionales con técnicas de descripción formal.* España, Universidad de Vigo, Departamento de Tecnologías de las Comunicaciones, ETSI de Telecomunicación, 2000. p.

DYKSTRA, J. *"Verificación y Validación de Software con Destiny: Un enfoque paralelo a la prueba automática de teoremas". En sitio: "ACM: Crossroads Student Magazine"*, 2002. [2008]. Disponible en: http://www.acm.org/crossroads/espanol/xrds83/destiny.htmlhttp://www.acm.org/

ESCALONA, M. J. K., 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.* , 2000. [Disponible en: <http://www.pst.informatik.unimuenchen.de/personen/kochn/ideas03escalonakoch.pdf>

ESTEREL. *Esterel*, 2008. [Disponible en: http://www-

sop.inria.fr/esterel.org/files/Html/About/AboutEsterel.htm

FERNÁNDEZ CARRASCO, O. M., GARCÍA LEÓN, DELBA Y BELTRÁN BENAVIDES, ALFA. *Un enfoque actual sobre la calidad del software.*, 1995. p.

FRANCISCO J., S. D., DIEGO J. BODAS SAGI, VALENTÍN POZO LLORENTE. *"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.*, 2005.

FRANCISCO JOSÉ GALÁN MORILLO, J. M. C. V. *Métodos Formales Orientados a Objetos*: 20.

FUNES, A. D., ARÍSTIDES Integración de Modelos en UML y Especificaciones Formales: Transformaciones de OCL a RSL, 2003.

FURIA, C. A. *TRIO: formal language, method and tools*, 2005. [2008]. Disponible en: http://risorse.dei.polimi.it/TRIO/

GALLARDO, M., MARÍA, DEL MAR; MERINO, GÓMEZ, PEDRO "Métodos para la Construcción de Software Fiable: Interpretación Abstracta", 2006.

GALLUD, J. A. G. C., J. M. "NOTOS: Un Generador de Prototipos para Especificaciones LOTOS" DISCA, 2007.

GARCÍA, M. B. *El proyecto informático de construcción del software.*, ? [2008]. Disponible en: http://cv.uoc.es/cdocent/TN51WR4HLVHTWDZD94AZ.pdf

GIL, A. S. *Diseño y verificación de sistemas distribuidos mediante la aplicación combinada de métodos formales.* . España, Universidad de Vigo, 1999. p.

GOGUEN, J. *"The OBJ Family".Sitio del Departamento de Ciencia de la Computación e Ingeniería. Universidad de California. San Diego, USA.*, 2005.

GÓMEZ, L., PRISCILA, RUTH, HUERTA,DEL JUNCAL ,JORGE LUIS *Herramientas Case*, 2008.

GONZALEZ, C. *Conceptos generales de calidad total*, 2007. [Disponible en: http://www.monografias.com/trabajos11/conge/conge.shtml

GONZALEZ URMACHEA, M. *"Redes de Petri"* 2008]. Disponible en: http://www.monografias.com/trabajos14/redesdepetri/redesdepetri.shtml#mo

GORM LARSEN, P. *"The VDM-SL Examples Repository"*, 2006.

GRENOBLE. *"Outils industriels / Industrial Tools". En sitio: "Grenoble B Site"*? [2008]. Disponible en: http://wwwlsr.imag.fr/B/Bsitepages.html

HANNEMAN., R. A. *INTRODUCCIÓN A LOS MÉTODOS DEL ANÁLISIS DE REDES SOCIALES?* : 6.

HIROTAKA TAKEUCHI, I. N. *Scrum*, 2008. [2008]. Disponible en: http://es.wikipedia.org/wiki/Scrum

IEEE. *Guide for Software Quality Assurance Planning*, 1900.

IEEE. *Guide for Software Quality Assurance Plans*, 2006.

IEEE, S. C., Ed. *"IEEE Standard for Software Quality Assurance Plans"*, 1998.

INDUSTRIAL, I. U. T. and "Calidad Total-Reingeniería-Normas ISO 9000." 1999.

INFOCALIDAD. 2004. [Disponible en: http://www.infocalidad.net/gest_calidad_def/definicion.asp

INSTITUTO, W. "Norma de Calidad. Gestión de La Calidad o Excelencia", 2006.

ISO/IEC. *Technology - Software product quality*, 2005.

I-SOL. *Cómo obtener un Software con Calidad*, 1999. [Disponible en: http://www.i-sol.com.ar/pg005.html

J. MCCALL, P. R., G.WALTERS. *Factors in Software Quality*, 1977. 3 vols.

JURISTO, N. M., ANA M.; VEGAS, SIRA "Técnicas de evaluación del software", 2006.

LARCH. *Larch Home Page*, 2001. [2008]. Disponible en: http://nms.lcs.mit.edu/Larch/

LARSEN, P. G. Overture - Open-source Tools for Formal Modeling, 2008.

LEAVENS, G. T. *"What good is it?" Departamento de Ciencias de la Computación de la Universidad del estado de Iowa.*, 1999.

LEON, F. *Métodos Formales*, 2006. 18.

LONDON, I. C. *LTSA Eclipse and WS-Engineer LTSA Eclipse y WS-Ingeniero* 2007. [2008]. Disponible en: http://translate.google.com.cu/translate?hl=es&sl=en&u=http://eclipse-plugins.info/eclipse/plugin_details.jsp%3Fid%3D1243&sa=X&oi=translate&resnum=1&ct=result&prev=/search%3Fq%3Dtool%2BLTSA%2Blicense%26hl%3Des%26sa%3DG o http://www.doc.ic.ac.uk/ltsa/eclipse/

LOTOS. *Lotos*, 1995. [2008]. Disponible en: http://wwwtios.cs.utwente.nl/lotos/

LOVELLE, J. M. C. *Calidad del software*, 1999. [2008]. Disponible en: www.uniovi.es

LUNA, C. D. *Enseñando Métodos Formales con Coq*, 2006. 10.

MACCOLL, I. C., DAVID. . *"Correctitud de la Interfaz con el usuario."* 1999. [2007]. Disponible en: http://www.acm.org/crossroads/espanol/xrds33/correct.html

MIGUEL, B. *"El proyecto informático de construcción de software."?* [Disponible en: http://cv.uoc.es/cdocent/TN51WR4HLVHTWDZD94AZ.pdf

MIRANDA, L. S. L. *"Sistema de Gestión de La Calidad"*, 2006. [2007]. Disponible en: http://www.gestiopolis.com/canales7/ger/sistemas de gestión de la calidad.htm

MUÑOZ, C. *"Métodos Formales, por ejemplo" Computer Science Laboratory, SRI International. En sitio: "NIA: National Institute of Aerospace"Publicado: 1998.* , 1998. [Disponible en: Disponible en:<http://www.csl.sri.com/users/munoz/Papers/mfhtml/mfhtml.html >

NASA. *"La computadora lo puede verificar" Publicado: 2003. En sitio: "NASA: National Aeoronautics and Space Administration"*, 2003. [2008]. Disponible en: http://www.nasa.gov/centers/ames/spanish/research/exploringtheuniverse/exploringtheuniverse computercheck.html

NAVARRO, A. *Revisión de técnicas formales*? : 26.

NÚÑEZ CAMALLEA, N. L. *Diccionario de Informática*, 2005.

PARADIGM, V. *Paradigma Visual para UML*, 2007. [2008]. Disponible en: http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(M%C3%8D)_14720_p/

PAREDES HERNÁNDEZ, C. U. *"Procesamiento Computacional del Lenguaje Natural" Tamaulipas, México*. México, 30/092000, 2000.

PASCUAL, J. S.; J. M. G. BARAHONA*, et al. Introducción al software libre*, 2003. [2008]. Disponible en: http://curso-sobre.berlios.de/introsobre/2.0.1/sobre.pdf

PAZOS ARIAS, J. J. *"Especificación Formal. Técnicas para la especificación no ambigua del software"* 2000. [2007]. Disponible en: http://wwwgris.det.uvigo.es/~jose/doctorado/fdt/sld001.htm

PEDERSEN, J. S. *RAISE - Rigorous Approach to Industrial Software Engineering*, 2004. [2008]. Disponible en: http://spd-web.terma.com/Projects/RAISE/

PIECE, K. *B-Toolkit*, 2002. [2008]. Disponible en: http://www.b-core.com/btoolkit.html

POLLAN CAÑETE, S. C. A., YAIMA. *Técnicas formales para el análisis y predicción del comportamiento fiable del software.* Cuba, Universidad de las Ciencias Informáticas, 2007. p.

REBOREDO, J. A. G. *Introducción a Spin*? : 19.

SÁNCHEZ PALMA, P. *"Animación de Especificaciones OASIS mediante redes de Petri Orientadas a Objeto" Tesis Doctoral de Informática. Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, España.* , 2000. [2008]. Disponible en: <http://www.dsic.upv.es/users/oom/docs/tesisppalma.pdf>

SANTIAGO, P., MARIO, DISTEFANO, ANTONIO, PASEO, ATILIO, RANZUGLIA UML y REDES DE PETRI EN LA EVALUACION DE PERFORMANCE DE SISTEMAS.

SOLLA, A. G. *Diseño y verificación de sistemas distribuidos mediante la aplicación combinada de métodos formales*, 1999. 256.

SOMMERVILLE, I. "Verificación y Validación" (Prentice Hall). 2000, 6 ta edición

S.PRESSMAN, R. *Ingeniería de Software. Un enfoque práctico.*, 1998. p.

TENEY, R. L. *"Estelle Information"* [online]. 2000. [Disponible en: Publicado: 25 abril de 2000. < http://www.estelle.org >

UML, A. *Argo UML*, 2007. [2008]. Disponible en: http://es.wikipedia.org/wiki/ArgoUML

UPM. *"Máster Europeo en Computación Lógica" Facultad de Informática de la Universidad Politécnica de Madrid (UPM).* 2004. [Disponible en: Disponible en: <http://www.clip.dia.fi.upm.es/mastercl/indice.html >

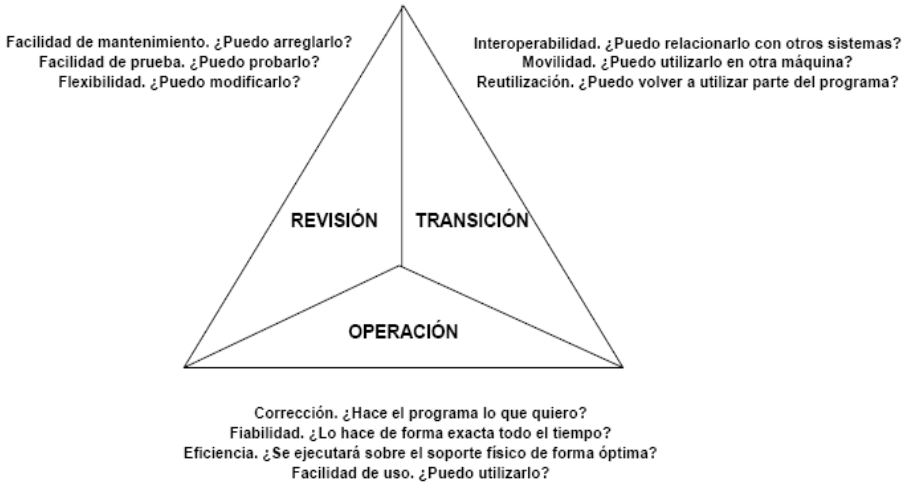
VAREA, M. *Especialización y Análisis Avanzados de Sistemas Pervasivos*, 2005. [2008]. Disponible en: http://users.ecs.soton.ac.uk/mv/research/asap.php?lang=es

VICENTE PELECHANO, O. P., EMILIO INSFRÁN, JOSÉ A. CARSÍ. *OO-Method: Una apuesta por la integración de técnicas formales y semi-formales en la ingeniería de requisitos*, 1996.

XLIRA. *XLira*? [Disponible en: http://tvdi.det.uvigo.es/proyectos/xlira/xlira.html

ANEXOS

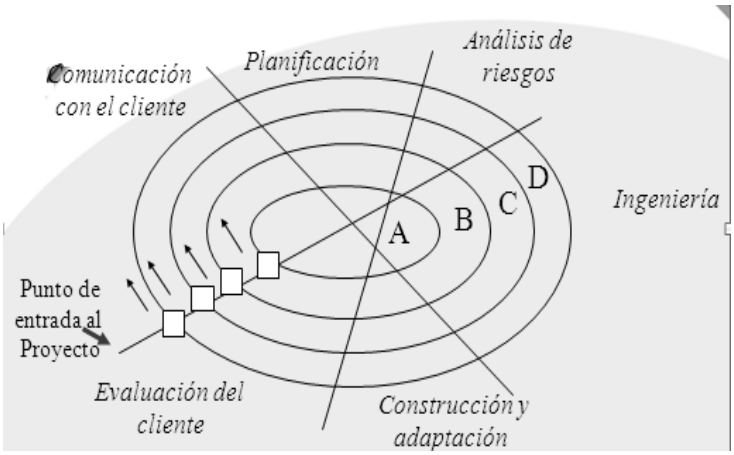
Anexo 1: Modelo de McCall



Anexo 2: Modelo de McCall

Factor	Criterio
Correctitud	✓Rastreabilidad ✓Completitud ✓Consistencia
Confiabilidad	✓Consistencia ✓Exactitud ✓Tolerancia a fallas
Eficiencia	✓Eficiencia de ejecución ✓Eficiencia de almacenamiento
Integridad	✓Control de acceso ✓Auditoría de acceso
Usabilidad	✓Operabilidad ✓Entrenamiento ✓Comunicación
Mantenibilidad	✓Simplicidad ✓Concreción

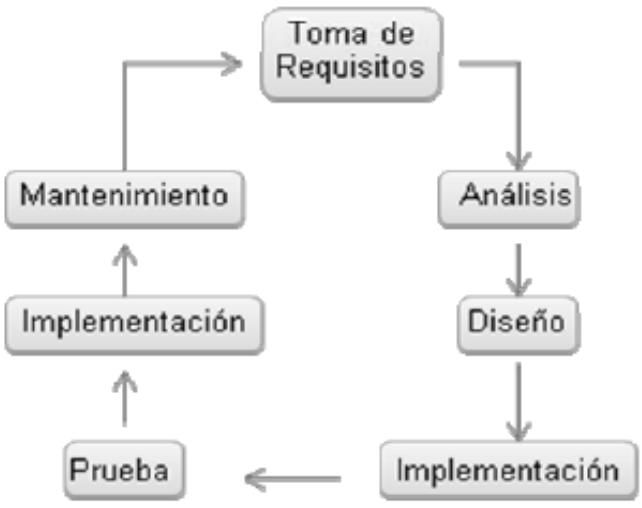
Anexo 3: Modelo espiral (Boehm)



A: Desarrolladores de Conceptos B: Desarrollo de Productos

C: Mejora de Productos D: Mantenimiento de Productos

Anexo 4: Ciclo de vida del software



Anexo 5: Encuesta a trabajadores de proyectos productivos de la UCI

Esta encuesta está destinada al personal vinculado con el proceso de desarrollo del software que laboran en los diferentes proyectos productivos de la Universidad. El objetivo de esta encuesta es conocer la forma en que se emplean los métodos en la producción de software para contribuir a la calidad de los proyectos de la UCI. Se darán las respuestas en dependencia de los conocimientos que tengan los encuestados y de acuerdo al roll que desarrollen. Si no está en condiciones de responder alguna pregunta, déjela en blanco.

Las preguntas son las siguientes:

1. ¿Tiene usted experiencia en la producción de software?

   si___ no___

2. ¿Cuál es el nombre de su proyecto?

   _____

3. ¿Cuál ROLL desarrolla usted en el proyecto?

   Líder___, Programador___, Analista___, Diseñador___, Arquitecto___

   Asesor de calidad___, Otros___Cuál_____

4. ¿Considera usted que el software producido en la Universidad de las Ciencias Informáticas cumple con las expectativas del cliente? ¿En qué porciento?

   100%___ 80%___ 50%___ 0%___

   Otro ___ Cuál ___

5. Valore la calidad de la producción de software en la Universidad de las Ciencias Informáticas.

   Buena___ Regular___ Poca___ Ninguna___

6. ¿Qué modelos del proceso de desarrollo de software usted conoce?

   ___Modelo en cascada o convencional

   ___Modelo evolutivo

   ___Modelo transformacional

   ___Modelo basado en reutilización

   ___Modelo en espiral

   ___Otros Cuales _____, _____,

   _____.

7. ¿Qué modelo del proceso de desarrollo de software se utiliza en su proyecto?

   _____

8. ¿En su proyecto se han realizado revisiones de calidad internas, es decir dentro del proyecto?

Si___ No___ No sé___

9. ¿Con qué frecuencia se han realizado dichas revisiones?

Varias veces___ Pocas veces___ Nunca___ No sé___

10. ¿En qué fase o fases del proceso de desarrollo del software se gestiona la calidad en su proyecto?

_____

11. ¿Cuáles modelos de calidad usted conoce que se implementan en la UCI?

_____,_____,_____

12. ¿Conoce usted los modelos de calidad que se implementan en su proyecto?

Si___ No___

¿Cuáles? _____,_____

_____,_____

13. ¿Sabe usted si a su proyecto se le han hecho revisiones externas de calidad?

Si___ No___ No sé___

14. ¿Con qué frecuencia se han realizado dichas revisiones?

Varias veces___ Pocas veces___ Nunca___ No sé___

15. ¿Qué actividades para el aseguramiento de la calidad del software se realizan en su proyecto?

__Métricas del software para el control del proyecto.

__Verificación y validación del software a lo largo del ciclo de vida. (Incluye las pruebas y los procesos de revisión e inspección).

__La gestión de la configuración del software

___ Otras Cuales _____, _____,_____

16. Diga en cada caso cuáles técnicas tradicionales y/o formales son usadas para el desarrollo de los productos en su proyecto en la etapa de ***especificación de requisitos***.

TÉCNICAS FORMALES	TÉCNICAS TRADICIONALES
_____	_____
_____	_____
_____	_____

¿Por qué medio obtuvo ese conocimiento?

Estudio Individual___ Preparación del proyecto___ Por casualidad___

17. Diga en cada caso cuáles técnicas tradicionales y/o formales son usadas para el desarrollo de los productos en su proyecto en la etapa de ***análisis y diseño*** del software.

TÉCNICAS FORMALES TÉCNICAS TRADICIONALES

_____ _____

_____ _____

_____ _____

¿Por qué medio obtuvo ese conocimiento?

Estudio Individual___ Preparación del proyecto___ Por casualidad___

18. Diga en cada caso cuáles técnicas tradicionales y/o formales son usadas cuando ***se programa*** el software en su proyecto.

TÉCNICAS FORMALES TÉCNICAS TRADICIONALES

_____ _____

_____ _____

_____ _____

¿Por qué medio obtuvo ese conocimiento?

Estudio Individual___ Preparación del proyecto___ Por casualidad___

19. ¿Cuándo se realizan las pruebas al producto que usted desarrolla?

Al finalizar___ Por etapas___ Al final de cada etapa___ Al inicio___

En todo el proceso___ Al inicio y final___ Nunca___ Otros momentos___

20. ¿Qué técnicas tradicionales y/o formales son usadas para probar el software?

TÉCNICAS FORMALES TÉCNICAS TRADICIONALES

_____ _____

_____ _____

_____ _____

¿Por qué medio obtuvo ese conocimiento?

Estudio Individual___ Preparación del proyecto___ Por casualidad___

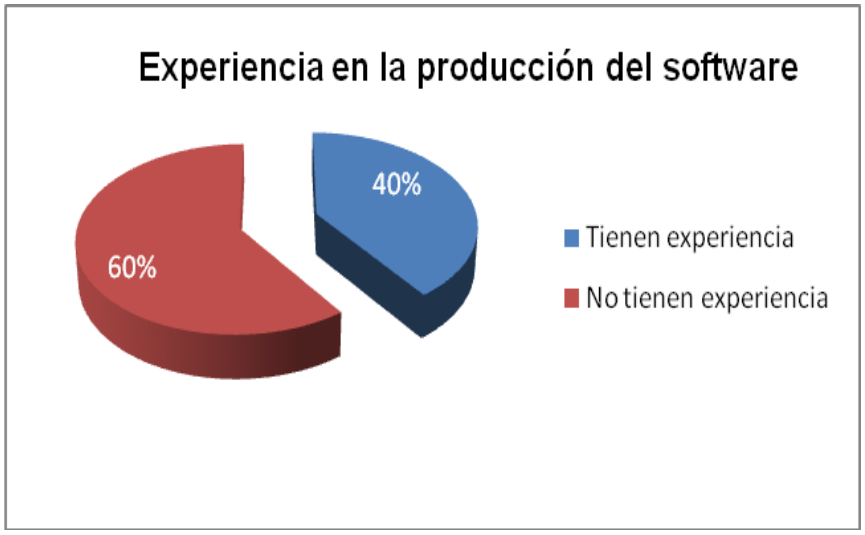
21. ¿Con qué frecuencia son usadas por usted las técnicas formales teniendo en cuenta su trabajo en el proyecto?

Muchas veces___ Siempre___ Algunas veces___ Poco___ Nunca___

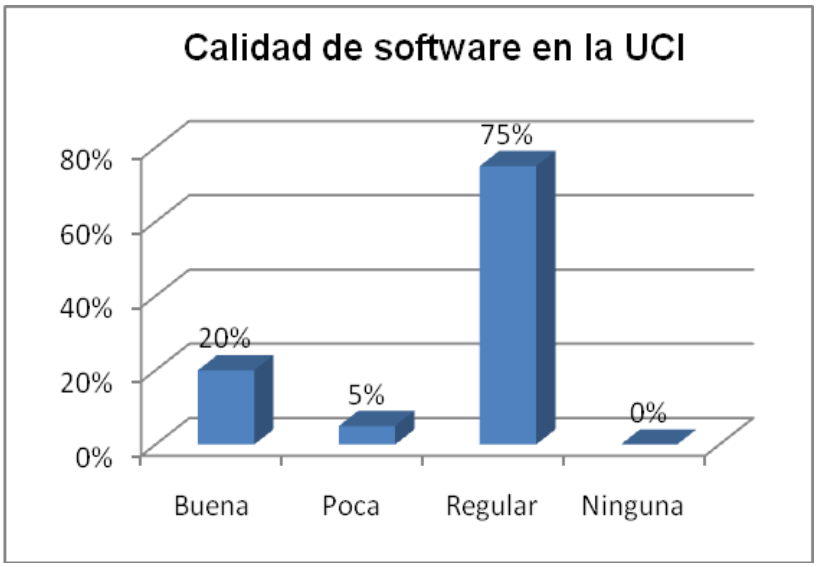
22. ¿Con qué frecuencia son usadas por usted las técnicas tradicionales teniendo en cuenta su trabajo en el proyecto?

Muchas veces___ Siempre___ Algunas veces___ Poco___ Nunca___

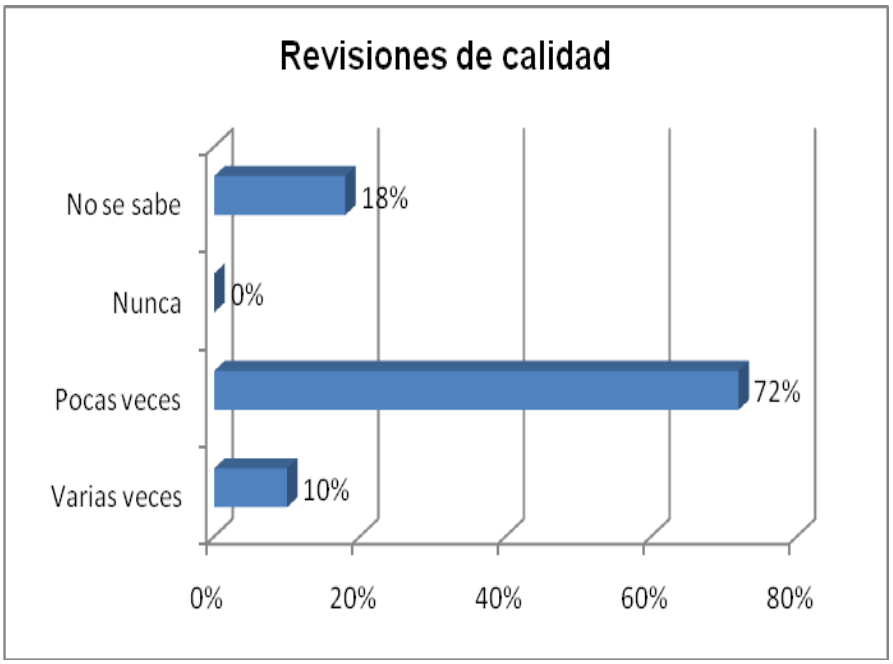
Anexo 6: Resultados de la encuesta



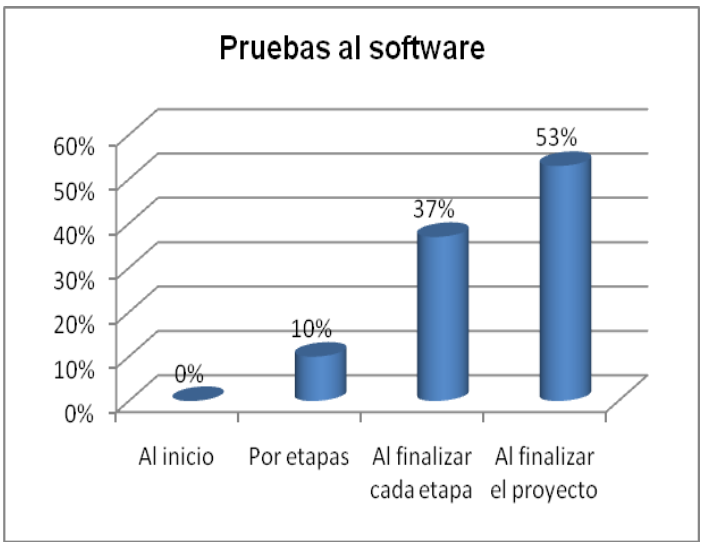
Anexo 7: Resultados de la encuesta



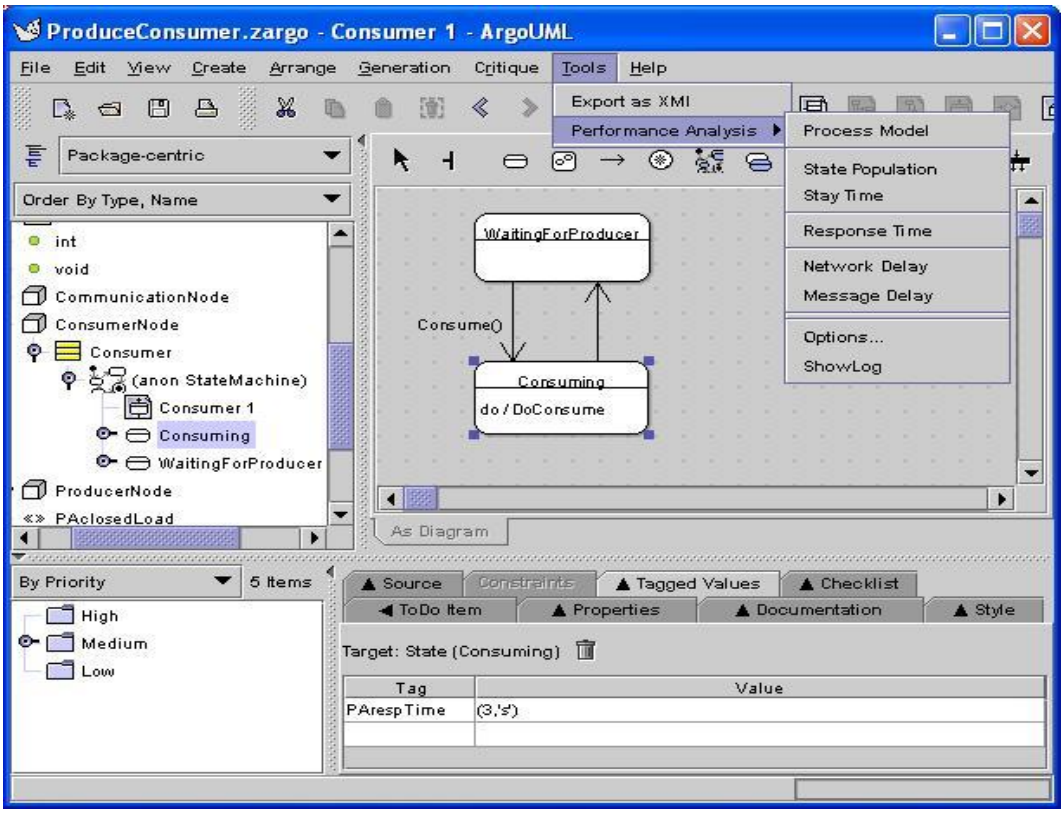
Anexo 8: Resultados de la encuesta



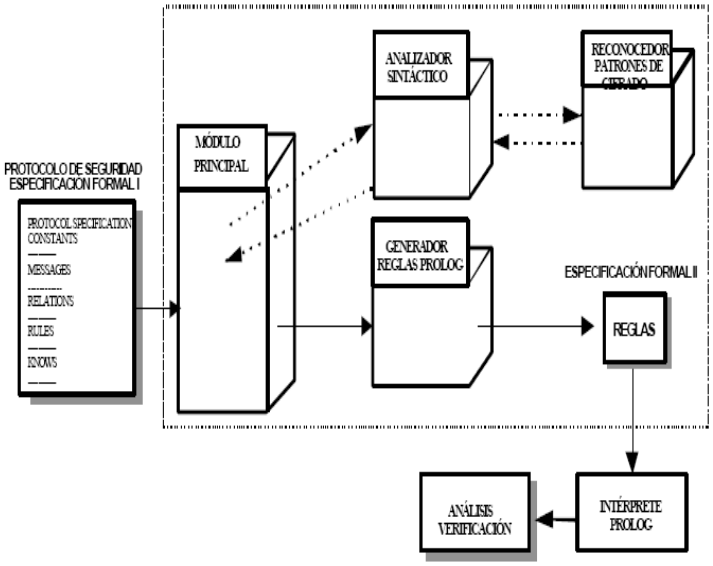
Anexo 9: Resultados de la encuesta



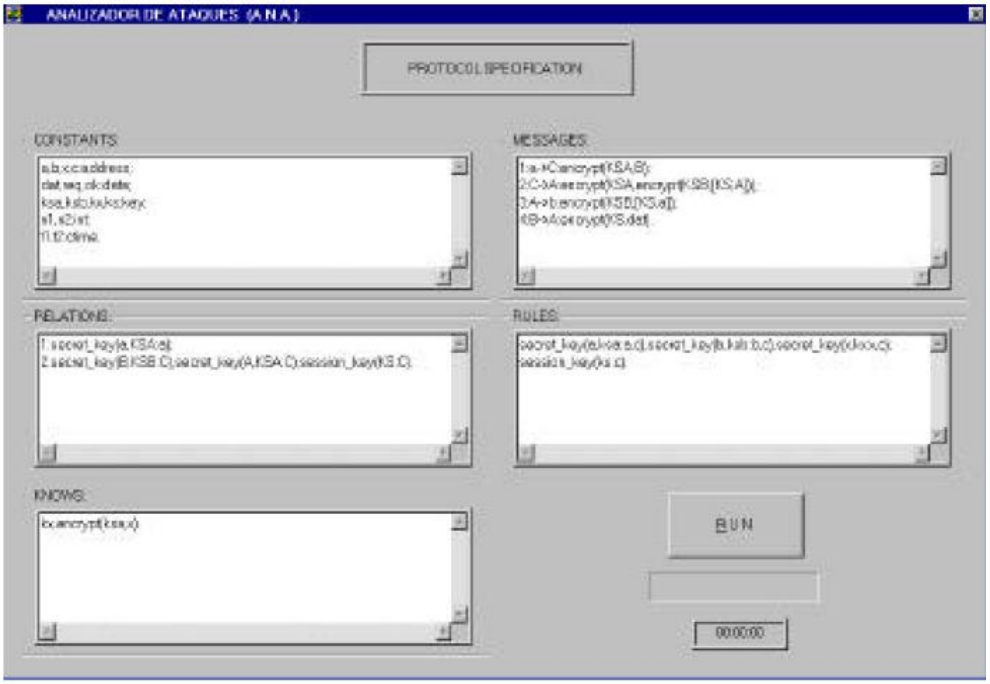
Anexo 10: Herramienta ArgoSPE



Anexo 11: Arquitectura lógica del Analizador de Ataques (A.N.A)



Anexo 12: Ejemplo de especificación de un protocolo de seguridad (especificación formal)



GLOSARIO DE TÉRMINOS

Automatizar: Mecanizar, motorizar.

Calidad (*Quality*)**:** Conjunto de propiedades y de características de un producto o servicio, que le confieren su aptitud para satisfacer unas necesidades explícitas e implícitas.

Calidad del producto: Es la resultante de una combinación de características de ingeniería y fabricación, determinante del grado de satisfacción que el producto proporcione al consumidor durante su uso.

Código: Es la sucesión de sentencias que dan lugar al programa fuente.

Componente: Que forma parte de alguna cosa o de su composición.

Cliente: Persona que demanda los servicios o productos que presta una Organización/Unidad Administrativa. Es el más próximo destinatario de los servicios o productos que ofrece una Unidad Administrativa.

Eclipse: Es una plataforma de software de Código abierto independiente de una plataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido".

Eficacia: Consiste en obtener el máximo resultado posible con unos recursos determinados, o en mantener con unos recursos mínimos la calidad y cantidad adecuada de un determinado servicio/producto. Se mide comparando los resultados realmente obtenidos con los previstos independientemente de los medios utilizados. También puede entenderse como la comparación entre los resultados obtenidos y un óptimo posible.

Empresa: Entidad integrada por el capital y el trabajo, como factores de la producción, y dedicada a actividades industriales, mercantiles o de prestación de servicios con fines lucrativos.

Equipo de desarrollo: Grupo de personas que se dedican a la realización de un producto determinado.

Gestión de calidad: Es un conjunto de actividades coordinadas para dirigir y controlar a un conjunto de personas e instalaciones con una disposición de responsabilidades, autoridades y relaciones en lo relativo a la Calidad.

Hardware: Conjunto de componentes materiales de un sistema informático. Cada una de las partes físicas que forman un ordenador, incluidos sus periféricos.

Herramienta: Es un dispositivo artificial cuya función es facilitar la aplicación de energía a una pieza o material durante la realización de una tarea. Es frecuente usar el término herramienta, por extensión, para denominar dispositivos o procedimientos que aumentan la capacidad de hacer ciertas tareas. Tal es el caso de herramientas de programación, herramientas matemáticas o herramientas de gestión. Esto frecuentemente viola la característica básica de las herramientas de ser medios para la aplicación controlada de energía.

IEEE: Corresponde a las siglas de *The Institute of Electrical and Electronics Engineers*, creada en 1884. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, ingenieros en sistemas e ingenieros en telecomunicación.

Implementar: Implantar, poner en marcha una instalación informática.

Iterar: Repetir

Lenguaje natural: Es el lenguaje hablado y/o escrito y/o signado por humanos para propósitos generales de comunicación, para distinguirlo de otros como puedan ser una lengua construida, los lenguajes de programación o los lenguajes usados en el estudio de la lógica formal, especialmente la lógica matemática.

Modelo: Cosa que ha de servir de objeto de imitación. Objeto, construcción u otra cosa con un diseño del que se reproduce más iguales. Esquema teórico de un sistema o de una realidad compleja que se elabora para facilitar su comprensión y estudio.

On-line: Indica que la aplicación o el sistema al que nos referimos permanece conectado a otro ordenador o a una red de ordenadores.

Proyecto: Proceso único consistente en un conjunto de actividades coordinadas y controladas con fechas de inicio y de finalización, llevadas a cabo para lograr un objetivo conforme con los requisitos específicos, incluyendo las limitaciones de tiempo, costo y recursos.

Producto: Lo que se produce o elabora. Beneficio o ganancia.

Proceso: (del latín *processuss*), es un conjunto de actividades o eventos que se realizan o suceden con un determinado fin. Este término tiene significados diferentes según la rama de la ciencia o la técnica en que se utilice.

Proyecto Productivo: Elemento organizativo a través del cual se gestiona el desarrollo de software. El resultado de un proyecto es una versión de un producto.

Satisfacción del cliente: Percepción del cliente sobre el grado en que se han cumplido sus requisitos.

Software: Todos los componentes intangibles de un ordenador o computadora, es decir, conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica, en contraposición a los componentes físicos del sistema (hardware). Esto incluye aplicaciones informáticas tales como un procesador de textos, que permite al usuario realizar una tarea.

Técnicas: Es un procedimiento o conjunto de procedimientos que tienen como objetivo obtener un resultado determinado, ya sea en el campo de la ciencia, de la tecnología, del arte o en cualquier otra actividad.

Usuario: Es la persona que utiliza o trabaja con algún objeto o que proviene de algún servicio público o privado, empresarial o profesional.