Universidad de las Ciencias Informáticas
Facultad 3

Título: Propuesta de un Sistema de Métodos Formales para la Especificación de Requisitos en los proyectos de la facultad 3.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

Autor(es): Yanelkys Pérez Vieyto.

Mindrey Gamboa Matos

Tutor(es): Miguel Ángel Martínez Acosta.

Consultantes: Karina Pérez Teruel.

Maikel Yelandi Leyva Vázquez.

Junio 2007.

DECLARACIÓN DE AUTORÍA

Declaramos que somos las únicas autoras de este trabajo y autorizamos al Área de Producción de la Universidad de las Ciencias Informáticas, así como dicho centro a hacer uso del mismo en su beneficio.

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

Yanelkys Pérez Vieyto

_____

Firma del autor

Mindrey Gamboa Matos

_____

Firma del autor

Ing. Miguel Ángel Martínez Acosta

_____

Firma del Tutor

"A veces sentimos que lo que hacemos es tan solo una gota en el mar,
pero el mar sería menos si le faltara una gota."

Agnes Gonsha Bojaxhiu

AGRADECIMIENTOS.

Queremos agradecer a todas las personas que hicieron posible la realización de este trabajo de diploma.

A nuestro tutor Miguel Ángel que nos guió y nos ayudo.

A los profesores Maikel Yelandi y Karina por dedicarnos parte de su tiempo guiándonos y aconsejando con sus criterios tan valiosos.

A nuestros compañeros, profesores y amigos que dieron junto a nosotros los primeros pasos en la universidad y a los que nos encontramos en el transcurso de este camino.
A nuestros seres queridos por su apoyo incondicional.

A nuestra Revolución y a nuestro comandante en jefe Fidel Castro Ruz por darnos la posibilidad de estudiar y graduarnos, por sus reflexiones que nos hace ser mejores personas.

DEDICATORIA.

Mindrey

A mi padre, Orlando, por su apoyo y enseñanzas, a mi madre, Merlis, por su amor y dedicación, por ser mis amigos, por todos los sacrificios que no han sido ni serán en vano. Los admiro y respecto, gracias por ayudarme a ser quien soy.

A mis dos tesoros, mis abuelos, Libia y Inocencio, gracias por mimarme y hacerme sentir la niña de sus ojos.

A mis tíos, en especial a mi tía María, gracias por estar ahí siempre que te necesite y darme tu apoyo incondicional.

A mi hermanito Orlandys, por impulsarme a enfrentar lo descocido, por aconsejarme, por ser la razón de luchar cuando creía que ya no me quedaban fuerzas, por ser el mejor regalo que me ha dado la vida.

A mi gran amor, Wiliam, por su apoyo, paciencia, consejos y por todos los momentos de alegrías y todos los que vendrán. Su amor me hace ser una mejor persona.

A mis amigos, ustedes saben quienes son, en especial a Utimia, por creer en mí, por escucharme y darme su apoyo siempre que lo necesité. Gracias a todos por caminar todo este tiempo a mi lado, nada sería lo mismo sin ustedes.

A mis suegros, Nory y Nelson, por sus consejos, por su cariño, por darle la vida al hombre con quien quiero compartir mi vida.

Yanelkys

Dedico este Trabajo de Diploma a Dios que me dio vida, a mi mamita Adita por darme el ser y todo lo maravilloso que me ha pasado, por ser grande y justa y por ser la vida mía, por quererme y dedicarse incondicionalmente a mí y por ser la mejor madre y amiga y hermana del mundo, a mi papa Carlos porque es el mejor padre que pudiera desear, por dedicarme toda su vida y quererme tanto, a mi esposo Osnex por cuidarme, respetarme y quererme a su manera y ser mi papito

Dedico también este trabajo a mi profesor de matemáticas Carlos Monsibay donde quiera que dios lo tenga por ayudarme y siempre guiarme por el mejor camino y aunque no pudo verme formada, no importa, le dedico mi éxito.
A mis tíos, tías y demás familia que influyeron de una manera u otra en que me formara como ingeniera.

También a mis primos Yaridys, Odalis y Jorgito por haberme apoyado cuando más los necesité y siempre tenderme la mano en los momentos más duros.
A mi maestra de primaria Clara La Hoz por guiarme siempre el caminos del saber y por se tan buena y paciente conmigo.

Y dedico este trabajo de diploma a mi Comandante en Jefe Fidel Castro Ruz, amigo, compañero y padre por permitirme escalar peldaño a peldaño el camino de la sabiduría y por luchar con tanto amor y valor por la Revolución que hoy nos entrega todo, por ser la luz de la esperanza del mundo.

RESUMEN.

Este trabajo de diploma se ubica en el campo de la Ingeniería de Software, disciplina que tiene como objetivo facilitar el desarrollo de productos software de calidad, que satisfagan las necesidades de los clientes. Uno de los factores que inciden en el cumplimiento de este objetivo es contar con una buena especificación de requisitos que posibilitaría que el grado de errores introducidos sea mucho menor durante el ciclo de vida del software. Una de las técnicas que se pueden aplicar para obtener una buena especificación de requisitos son los Métodos Formales y este trabajo se enmarca en el estudio de los mismos para proponer un Sistema de Métodos Formales que mejore la Especificación de los Requisitos en la facultad 3.

Los métodos formales aplicados en la especificación de requisitos producen resultados más precisos que el uso de los lenguajes naturales, ya que estos permiten especificar los requisitos de forma tal que eliminen ambigüedades surgidas del resultado de aplicar métodos convencionales. Además la especificación de estos tendría características tales como completitud y consistencia. Lo cual no quiere decir que la aplicación de los mismos no presente limitaciones.

Esta investigación propone un Sistema de Métodos Formales para la Especificación de Requisitos utilizando OCL como lenguaje base para la especificación y agregando Z y Lotos para la especificación de requisitos críticos.

PALABRAS CLAVE.
Ingeniería de Requisitos, Lenguaje Formal, Especificación Formal, Métodos Formales.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN.

La historia del Software está llena de proyectos arruinados ya sea por una planificación irreal, mala calidad del trabajo, personal inadecuado, cambios no controlados y funcionalidades que no satisfacen las necesidades de los usuarios. Desde que la Ingeniería del Software apareció a finales de los años 60, su aplicación ha permitido gestionar las necesidades del proyecto en forma estructurada, mejorar la capacidad de predecir cronogramas de proyectos, disminuir los costos y retrasos del mismo, mejorar la calidad del software y la comunicación entre equipos, evitando rechazos de usuarios finales.

La Ingeniería de Software tiene, como uno de sus grandes retos, producir software que realmente cumpla con lo que el cliente desea. A pesar que sus avances se dejan ver con frecuencia en estos últimos 30 años, en el desarrollo de técnicas y metodologías para lograr mejores resultados en las diversas etapas por la que pasa un software, el índice de proyectos no cumple con todas las expectativas del cliente. Se considera que una de las causas por lo que ocurre este hecho es por la poca habilidad que hay para producir requisitos de software que sean correctos, completos y sin ambigüedad.

La ingeniería de requisito surge para darle solución al problema antes planteado, es la primera fase por la que pasa un producto software y a la vez una de la más importante, porque en ella se determinan y se plasman las necesidades del cliente, se tratan los principios, métodos, técnicas y herramientas que permiten descubrir, documentar y mantener los requisitos para sistemas basados en computadora, de forma constante. La misma consiste en un conjunto de actividades y transformaciones que pretenden comprender e identificar el propósito del sistema y el contexto en el que será usado; donde los requisitos son el puente de enlace entre las verdaderas necesidades de los clientes, usuarios y otros vinculados al sistema.

Según el estándar IEEE 610.12-1990 un requisito se define como una condición o capacidad que un sistema software debe alcanzar para satisfacer un contrato, estándar, especificación o cualquier otro documento formal que se imponga (necesidades del cliente).

Además los requisitos deben ser claros ya que definen qué es lo que el sistema debe hacer, para esto se identifican las funcionalidades requeridas y las restricciones que se imponen, es una característica que un sistema debe tener para cubrir alguna de las necesidades que lo motivan.

La IEEE Standard Glossary of Software Engineering Terminology define un requerimiento como:

- Condición o capacidad que necesita un usuario para resolver un problema en lograr un objetivo.
- Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.
- Una representación documentada de una condición o capacidad en 1 o 2.

La ingeniería de requisitos proporciona el mecanismo adecuado para entender y valorar lo que el cliente necesita, negociando razonablemente, especificando una solución sin ambigüedades, validando la misma y gestionando los requisitos para que se transformen en un sistema operacional.

El proceso de ingeniería de requisito puede ser descrito en 5 pasos distintos:(PRESSMAN 2005)

- Identificación de Requisitos.
- Análisis de Requisitos y Negociación.
- Especificación de Requisitos.
- Modelizado del Sistema.
- Validación y Gestión de Requisitos.

La Identificación de Requisitos es el intercambio con el cliente, los usuarios y los que están involucrados en los objetivos del sistema o producto que se quiere realizar, para investigar como

los mismos se ajustan a las necesidades del negocio, finalmente como van a ser utilizados a diario.

El Análisis de Requisitos y negociación es la fase que va después de ser recopilados los requisitos, los mismos se agrupan por categorías y se organizan en subconjuntos, se estudia cada requisito en relación con el resto, se examinan en su consistencia completitud y ambigüedad y se clasifican en base de los clientes o usuarios.

La Especificación de Requisitos establece las metas y objetivos del software describiéndolo en el contexto del sistema basado en computadoras.

El Modelizado del Sistema es hacer un modelo que facilite toda la información de lo que se va a desarrollar. Con esto será más fácil asegurar la eficiencia del trabajo.

La Validación de Requisitos es la encargada de examinar las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencia, sin omisiones, que los errores detectados hayan sido corregidos y que el resultado del trabajo se ajusta a los estándares establecidos.

La Gestión de Requisitos es el conjunto de actividades que ayudan al equipo de trabajo a identificar, controlar y seguir los requisitos y los cambios en cualquier momento.

Este trabajo se centrará en la especificación de requisitos la cual establece los requerimientos y restricciones del sistema. Esta le permite al cliente describir de manera precisa qué es lo que quiere obtener y a lo desarrolladores, comprender qué es lo que quiere el cliente.

La especificación de requisitos tiene como objetivos:

- Proporcionar una base para estimaciones
- Proporcionar un “contrato” para validación y verificación
- Definir un documento base para futuras versiones o ampliaciones
- Establecer un punto de inicio para la comprensión del proceso de desarrollo.

Basado en el estándar IEEE Std. 830:1998: Práctica Recomendada para la Especificación de Requisitos Software: Un documento de especificación de requisitos debe tener las siguientes características:

- Correcto.
- No ambiguo.
- Completo.
- Consistente.
- Verificable.

Correcto: Un documento de especificación de requisitos es correcto sí y sólo sí cada requerimiento que está en él representa algo requerido por el sistema a construirse.

No ambiguo: Un documento de especificación de requisitos es no ambiguo sí y sólo sí cada requerimiento establecido tiene una única interpretación.

Completo: Un documento de especificación de requisitos es completo si posee cuatro cualidades:

- Todo lo que el software se supone que debe hacer esta incluido en el documento de especificación de requisitos. El sistema debe hacer las cosas establecidas en este documento de especificación de requisitos y nada más.

- Para toda entrada mencionada en el documento de especificación de requisitos, este detalla cual es la salida apropiada.

- Todas las páginas están numeradas con nombres y referenciadas; todos los términos y unidades de medida son proveídos; y todos los materiales y secciones mencionados están presentes.

- Insertar las letras "TBD" (To be determined, pendiente de ser determinado") en una sección de un documento de especificación de requisitos debe se evitado en la medida de lo posible.

Consistente: Un documento de especificación de requisitos es consistente sí y solo sí ningún subconjunto de los requerimientos individuales establecidos en él tienen conflictos.

Verificable: Un documento de especificación de requisitos es verificable sí y solo sí cada uno de los requerimientos establecidos en él es verificable. Un requerimiento es verificable si y solo si existe algún proceso costo efectivo en el cual una persona o máquina puede chequear que el software actual "como esta construido" satisface ese requerimiento.

Los requisitos del software contenidos en la especificación son más detallados, lo que conlleva al éxito de la implementación del software, el ambiente de la especificación de estos Requisitos debe de estar descrito de manera tal que no detalle aspectos del área de diseño o de implementación.

La Especificación de Requisitos del Software se somete a una revisión que puede ser llevada a cabo por el desarrollador o por el cliente, inicicialmente se realiza a nivel microscópico, en el que los revisores intentan asegurarse de que la especificación sea completa, consistente y correcta.

De la misma manera se hace una exploración completa de cada uno de estos dominios, la revisión es profunda y busca el detalle, examinando no solo las descripciones superficiales, sino la vía en la que los requisitos son expresados. La calidad de la Especificación de Requisitos Software es muy difícil de medir, la misma no garantiza que no tenga problemas.

De lo antes expuesto se deriva la siguiente **Situación Problémica**:
La definición de las funcionalidades de un sistema es un proceso complejo, pues él debe cumplir con los requisitos especificados para satisfacer las necesidades de los usuarios finales o de los clientes. Los proyectos de la Facultad 3 no realizan una buena Especificación de Requisitos, se basan solo en un lenguaje natural, lo que conlleva a que:

- Se obtengan requisitos poco consistentes.
- No se transformen los requisitos después de capturados en manejables y analizables.
- Existan Áreas no especificadas y Requisitos contradictorios,
- Afirmaciones (aparentemente) vagas e irrelevantes
- La especificación de la aplicación es simplemente textual.

Para realizar este proceso, no existe una única técnica estandarizada y estructurada que ofrezca un marco de desarrollo que garantice la calidad del resultado, lo que conlleva a un atraso circunstancial del trabajo a realizar.

Que da lugar al **Problema Científico:**
Las deficiencias en el proceso de especificación de requisitos en los proyectos de la facultad 3, afectan el proceso de desarrollo del software y la calidad del producto final.

Donde se determina como **Objeto de estudio**:
Métodos formales para la especificación de requisitos.

Objetivo de investigación:
Elaborar un Sistema de Métodos Formales para la especificación de requisitos en los proyectos de la facultad 3.

Y como **Campo de acción**:
La especificación de requisitos en los proyectos de la facultad 3.

A partir de lo anteriormente expuesto se plantea la siguiente **Hipótesis**:

Si se aplican Métodos Formales para la especificación de requisitos que se realiza en los proyectos de la facultad 3 se podrán corregir muchos de los errores que aparecen en el proceso de especificación de requisitos mejorando así la calidad del producto final.

Teniendo como **Variables Independientes**: Proceso de especificación de requisitos

Y como **Variables Dependientes**:
Proceso de desarrollo del software.
La calidad del Software.

Las Tareas de investigación:

1. Estudio de la especificación de requisitos.
2. Investigación sobre los diferentes tipos de Métodos Formales que se aplican a la especificación de requisitos.
3. Comparación de los diferentes tipos de Métodos Formales que se aplican a la especificación de requisitos.

Para hacer más segura esta investigación se emplearon métodos teóricos y empíricos, en función del objeto que se va a investigar.

Nivel teórico.

Análisis y síntesis: Se lleva a cabo para realizar un análisis de los Métodos Formales obteniendo los elementos de mayor importancia para poder elaborar los fundamentos teóricos de la investigación.

Inducción – Deducción: Se crean suposiciones a partir del conocimiento que ya se tiene referente a los Métodos Formales para la especificación de requisitos y se realizan comparaciones con el estudio bibliográfico, llegando así a una o varias generalizaciones.

Con el marcado objetivo de darle solución a la situación antes planteada se ha desarrollado un sistema métodos formales que garantizan la efectividad del producto.

La presente tesis esta compuesta por 3 capítulos:

CAPITULO 1: Fundamentación Teórica.

En este capítulo se tratan los principales conceptos a los que se hace alusión mas adelante, aplicaciones generales de los métodos formales que se aplican a la especificación de requisitos, así como las herramientas que utilizan y la importancia de los mismos.

CAPITULO 2: Estudio de los Métodos Formales para La Especificación de Requisitos.

En este capítulo se tratará de manera profunda y abarcadora sobre los distintos métodos formales, encontrados, aplicados a la especificación de requisitos permitiendo sacar conclusiones al respecto.

CAPITULO 3: Sistema de Métodos Formales para la Especificación de Requisitos.

En este capítulo ya se da la propuesta fundamentada, así como las herramientas a usar para dar solución a la problemática planteada y las opiniones de especialistas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

Introducción

En este capítulo se estudian los fundamentos teóricos relacionados con los Métodos Formales y el papel que juegan en la ingeniería del software. Se lleva a cabo un estudio de los proyectos de la facultad 3 y enmarcándose en estos, se continúa con la investigación. Se abordará además donde se han aplicado los Métodos Formales de manera general y de esta forma la veracidad de su importancia, así como las dificultades que estos presentan.

1.1 Situación en la Facultad 3.

Al realizar una investigación de cada proyecto y entrevistando al Vice-decano de Producción de la Facultad 3, se determinó que en la misma no se utilizan los métodos formales en las etapas del desarrollo del software y mucho menos en la Especificación de Requisitos, además de tener total desconocimiento de los mismos. Los integrantes de estos proyectos se limitan nada más a capturar los requerimientos y seguido de esto, redactan el documento de especificación de requisitos donde los clasifican y hasta ahí llegan en esta fase utilizando solo el lenguaje natural, lo que no le da seguridad de que el sistema este libre de errores. Esto conllevó al estudio de los Métodos Formales, tema avanzado de la ingeniería del software.

Según la 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”.

1.2 Un poco de historia.

En la década de los 40 surgieron los Lenguajes Formales debido a la introducción de los programas en la memoria principal, seguido de estos programas los lenguajes de programación

de alto nivel. En 1967 se introdujeron los métodos formales, mediante el concepto para programas secuenciales y un método basado en pre-condiciones, post-condiciones, y condiciones de terminación. Apoyado en este método, en ese mismo año se construye una lógica, y se define una serie de reglas de inferencia para poder reducir la demostración de la corrección de un programa a la demostración de la corrección de cada una de las sentencias que lo componen.

El primer avance serio que se realizó sobre los dos métodos anteriores lo introdujo (PNUELI 1977) con el uso de la lógica temporal para razonar sobre programas concurrentes. (CHANDY 1992) proponen un método para razonar sobre programas concurrentes en sistemas abiertos, y que aplican para los programas desarrollados con el lenguaje composicional CC++, también basado en representar el comportamiento de los programas como secuencias de estados.

Lo antes expuesto es a modo de historia de los métodos formales para brindar cierta ubicación en lo primeros indicios de estos, no significa que aquí terminó porque hay personas que incursionaron en este tema e introdujeron otros métodos e incluso hicieron combinaciones entre estos. No se puede hablar de un método formal si no se ha hablado de un lenguaje formal que fue quién da paso al desarrollo de los mismos.

1.3 Lenguaje Formal.

Los lenguajes formales cuentan con reglas semánticas y sintácticas rígidas, concretas y bien definidas de los lenguajes naturales como el inglés, donde la sintaxis y la semántica no se pueden controlar fácilmente. Los intentos de formalizar los lenguajes naturales, lleva a la construcción de gramáticas, como una forma de describir estos lenguajes. Entonces se puede caracterizar un lenguaje mediante las reglas de una gramática adecuada.

Lenguaje de Especificación Formal.

Un lenguaje de especificación formal está constituido por tres componentes primarios:(PRESSMAN 2006)

- Una sintaxis que define la notación científica con la cual se representa la especificación.
- Una semántica que ayuda a definir que se utilizará para describir el sistema.
- Conjunto de relaciones que definen las reglas que indican cuales son los objetivos que satisfacen la especificación.

Los lenguajes de especificación formal, posee tanto un dominio semántico como uno sintáctico. El dominio sintáctico utiliza una simbología que sigue estrechamente a la notación de conjuntos y al cálculo de predicados. El dominio semántico capacita al lenguaje para expresar los requisitos de forma concisa.

Especificación Formal.

La especificación formal es un área de investigación cuyo propósito es el desarrollo de técnicas, lenguajes y herramientas (basadas en lógicas clásicas y no clásicas, álgebras o cálculos) para alcanzar una de las principales metas de la ingeniería de software: permitir la construcción de sistemas que operen confiablemente a pesar de su complejidad. Es una descripción abstracta con una semántica formal orientada a modelos y orientada a propiedades.

Ya hecho todo este bosquejo se puede ver que es un Método Formal y demás desempeños.

1.4 ¿Qué son los Métodos Formales?

“Los métodos formales permiten al ingeniero de software crear una especificación sin ambigüedades que sea más completa y constante que las que se utilizan en los métodos convencionales u orientados a objetos. La teoría de conjuntos y las notaciones lógicas se utilizan para crear una sentencia clara de hechos (o de requisitos). Esta especificación matemática entonces se puede analizar para comprobar que sea correcta y constante. Como esta

especificación se crea utilizando notaciones matemáticas, inherentemente es menos ambigua que los nodos informales de presentación." (PRESSMAN 2002)

Según Steve Esterbrook:"Los métodos formales tienen un potencial tremendo para mejorar la claridad y la precisión de las especificaciones de los requisitos y a la hora de encontrar tanto errores importantes como sutiles."

Anthony Hall plantea que "Los Métodos Formales son objeto de controversia quienes los propugnan afirman que pueden revolucionar el desarrollo del software. Sus detractores piensan que resultan imposiblemente difíciles. Mientras tanto, para la mayoría de la gente son tan poco familiares que resulta difícil juzgar estos puntos de vista contrapuestos."

La Encyclopedia of software Engineering define los métodos formales de la siguiente manera: Un método formal es formal si tiene sólidas bases matemática usualmente proporcionadas por un lenguaje de especificación. Esta Base ofrece los medios de definir con precisión nociones como consistencia y completud, especificación, implementación y corrección.

En términos de ecuación

Métodos Formales = lenguaje de especificación + razonamiento formal

Son Técnicas basadas en las matemáticas como lenguaje y en herramientas (no necesariamente automatizadas) de análisis. También se consideran Métodos, mecanismos y formalismos para modelar, analizar y sintetizar.

Los Métodos Formales están sustentados por sólidas bases como son:

- La notación y la heurística de conjunto.
- La especificación constructiva.
- Operadores de conjunto.
- Operadores lógicos y secuencias.

1.5 Conceptos subyacentes que rigen los métodos formales.

Consistente: Los hechos planteados no deben contradecirse en otro sitio.

Ambigüedad: Planteamientos que se interpretan de varias formas.

Invariante de Datos: Conjunto de condiciones verdaderas a lo largo de la ejecución del problema que contiene una colección de datos. Define lo que esta garantizado que no cambiará.

Precondición: Define circunstancias en las cuales es válida una operación particular.

Poscondición: Define lo que esta garantizado que será cierto hasta completar la condición.

Estado: Una representación del modo de comportamiento observable externamente de un sistema o los datos almacenados a los que el sistema tiene acceso y puede alterar.

Operación**:** Una acción que tiene lugar en un sistema y lee o escribe datos a un estado. Una operación asociada con condiciones (precondición y poscondición).

Semántica formal: Semántica basada en teoría de conjuntos álgebra, lógica, autómatas, teoría de grafos, etc.

Heurísticas: Son formas concretas de realizar tareas, o de resolver problemas, avaladas por la experiencia práctica, que normalmente, aunque no siempre, permiten obtener buenos resultados.

Evento: Es un suceso significativo que debe tenerse en cuenta ya que influye en el comportamiento y evolución del sistema, tiene lugar en un punto del tiempo y carece de duración respecto al grado temporal del sistema

1.6 Los Diez Mandamientos de los Métodos Formales.

La decisión de hacer uso de los métodos formales (PRESSMAN 2005) en el mundo real no debe adoptarse a la ligera. Bowen y Hinchley han acuñado los diez mandamientos de los métodos formales, como guía para aquellos que estén a punto de embarcarse en este importante enfoque de la ingeniería del software.

I. Seleccionarás la notación adecuada: Con objeto de seleccionar eficientemente dentro de la amplia gama de lenguajes de especificación formal existente, el ingeniero del software deberá considerar el vocabulario del lenguaje, el tipo de aplicación que haya que especificar y el grado de utilización del lenguaje.

II. Formalizarás, pero no de más: En general, resulta necesario aplicar los métodos formales a todos los aspectos de los sistemas de cierta envergadura. Aquellos componentes que sean críticos para la seguridad serán nuestras primeras opciones, e irán seguidos por aquellos componentes cuyo fallo no se pueda admitir (por razones de negocio).

III. Estimarás los costes: Los Métodos Formales tienen unos costes de arranque considerables. El entrenamiento del personal, la adquisición de herramientas de apoyo y la utilización de asesores bajo contrato dan lugar a unos costes elevados en la primera ocasión. Estos costes deben tenerse en cuanta cuando se este considerando el beneficio obtenido frente a esa inversión asociada a los métodos formales.

IV. Poseerás un experto en métodos formales a tu disposición: El entrenamiento de expertos y la asesoría continua son esenciales para el éxito cuando se utilizan métodos formales por primera vez.

V. No abandonarás tus métodos formales de desarrollo: Es posible, y en muchos casos resulta deseables integrar los métodos formales con los métodos convencionales y los métodos orientados a objetos. Cada uno de estos métodos posee sus ventajas y sus inconvenientes. Una combinación de ambos, aplicada de forma adecuada, puede producir excelentes resultados.

VI. Documentarás suficientemente: Los métodos formales proporcionan un método conciso, sin ambigüedades y consistente para documentar los requisitos del sistema. Sin embargo, se recomienda que se adjunte un comentario en lenguaje formal, para que sirva como mecanismo para reforzar la comprensión del sistema por parte de los lectores.

VII. No comprometerás los estándares de calidad: Los métodos formales no tienen nada de mágico y, por esta razón, las demás actividades de SQA deben de seguir aplicándose cuando se desarrollen sistemas.

VIII. No serás dogmático: El ingeniero se software debe reconocer que los métodos formales no son una garantía de corrección. Es posible (o como algunos dirían) que el sistema final, aun cuando se haya desarrollado empleado métodos formales, siga conteniendo pequeñas omisiones, errores de menor importancia y otros atributos que no satisfagan nuestras expectativas.

IX. Comprobarás, comprobarás y volverás a comprobar: Los métodos formales no absuelven al ingeniero del software de la necesidad de llevar a cabo unas comprobaciones exhaustivas y bien planeadas.

X. Reutilizaras cuanto puedas: A la larga, la única forma racional de reducir los costes del software y de incrementar la calidad del software pasa por la reutilización. Los métodos formales no modifican esta realidad. De hecho, quizás suceda que los métodos formales sean un enfoque adecuado cuando es preciso crear componentes para bibliotecas reutilizables

1.7 Técnicas de Descripción Formal.

En la actualidad se pueden considerar las Técnicas de Descripción Formal como una forma correcta de describir el comportamiento de sistemas reactivos de procesamiento de la información. Se utilizan para denominar cualquier técnica o método que permite definir de forma completa el comportamiento de un sistema (hardware o software) mediante un lenguaje con sintaxis y semántica formales.

Por lo que se puede plantear que una técnica de especificación formal es un método formal especialmente adecuado para escribir especificaciones de sistemas que cuenta con dos propiedades fundamentales como son:

Expresividad: Debe ser lo suficientemente expresivo para poder describir, de forma clara, el comportamiento de los sistemas que se quieren modelar.

Abstracción: Debe ser lo suficientemente abstracto para poder describir qué hace un sistema sin poner restricciones a cómo lo hace.

1.7.1 Las Técnicas de Descripción Formal más difundida.

Las Técnicas de Descripción Formal que más se conocen en la actualidad se dividen en cuatro categorías:(DUKE 2000)

1. Técnicas algebraicas: Donde un sistema se modela mediante álgebras multi-sorted como un conjunto de tipos y de operaciones sobre esos tipos. Cada tipo (short) tiene un conjunto de valores y un conjunto de 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.

Estos métodos son especialmente útiles para especificar interfaces entre los componentes del sistema, considerando en este nivel de abstracción, el sistema como un conjunto de tipos

abstractos de datos. Las técnicas algebraicas presentan problemas para la verificación formal de la completitud de las especificaciones.

2. Técnicas basadas en modelos matemáticos: Son basadas en teoría de conjuntos y lógica de primer orden. El espacio de estados del sistema se modela a través de sus componentes que son a su vez modelados mediante conjuntos, funciones, etc. Se pueden formular condiciones invariantes sobre el espacio de estados mediante predicados sobre sus componentes. Las operaciones que manipulan el espacio de estados se especifican mediante predicados que relacionan diferentes estados de los componentes.

La especificación explícita del espacio de estados y de operaciones sobre estados constituye la principal diferencia con las técnicas algebraicas, que no utilizan los estados para modelar un sistema sino tipos abstractos de datos y operaciones sobre tipos.

3. Técnicas basadas en álgebra de procesos: Las álgebras de procesos se especializan en modelar las interacciones entre procesos concurrentes. En este tipo de sistemas, propiedades como: seguridad (el sistema no hace nada malo) y viveza (finalmente algo bueno ocurriría) son muy importantes. Estas técnicas son especialmente útiles en la especificación de sistemas distribuidos y concurrentes, como son los protocolos y servicios de telecomunicaciones.

4. Técnicas basadas en lógica: Permiten especificar explícitamente las propiedades de seguridad y viveza de un sistema. Un sistema se especifica a través de un conjunto de fórmulas de la lógica que definen relaciones y sucesos que ocurren en el tiempo. Permiten trabajar con especificaciones parciales de un sistema.

Estas técnicas son muy apropiadas para la especificación de sistemas de tiempo real. Su principal inconveniente está en su incapacidad para expresar la estructura- arquitectura de un sistema, de ahí que se utilicen con más frecuencia en las primeras fases del diseño. De la

clasificación anterior se puede extraer como conclusión que ningún tipo de técnica se adapta perfectamente a todas las fases del diseño y desarrollo de un sistema concreto.

1.7.2 Otras Técnicas de Descripción Formal.

Existen otras técnicas de descripción formal en función de los aspectos del sistema que modela:

- Funcionalidad del sistema, estas son las de mayor peso.
- Propiedades concurrentes. De especial interés para la especificación de sistemas distribuidos y de comunicaciones.
- Tiempo. De especial interés para la especificación de sistemas en tiempo real. Están menos desarrolladas que las dos anteriores.

Técnicas de Descripción Formal en una clasificación muy extendida.

Otra clasificación muy extendida de las Técnicas de Descripción Formal define dos grandes grupos:(GOTZHEIN 1992)

- Constructivas. Las especificaciones escritas con estas técnicas son ejecutables. Su principal ventaja es su adecuación para realizar un prototipo rápido del sistema final. Su principal inconveniente es la imposibilidad de especificar de forma explícita las propiedades de un sistema, lo que impide la verificación explícita de propiedades.
- No constructivas u orientadas a propiedades. Las propiedades del sistema se expresan de forma explícita. Permiten tratar cada propiedad de forma separada lo que ayuda a decidir si es o no esencial. Su principal inconveniente reside en la dificultad para decidir si la especificación de un sistema es o no completa, es decir, si define únicamente comportamientos correctos.

Por tanto, las técnicas de descripción formal constructivas y no constructivas tienen características complementarias que sugieren combinar su empleo dentro del proceso de desarrollo: Las Técnicas de Descripción Formal orientadas a propiedades (no constructivas) en la fase inicial de especificación de requisitos y las constructivas en la fase de diseño.

1.8 Razones para utilizar Técnicas de Descripción Formal.

Las técnicas de Descripción formal son de gran importancia ya que contribuyen a que el sistema sea especificado con una mayor calidad, además:

- El desarrollo de especificaciones formales permite una mejor comprensión de los requisitos del sistema, reduciendo errores y omisiones en la definición de los mismos. Además, proporcionan la base para un buen diseño.
- Las especificaciones formales son entidades matemáticas que pueden analizarse empleando métodos matemáticos. Es posible demostrar la consistencia y completitud de una especificación, la adecuación entre una especificación y una implementación, y la ausencia de ciertos tipos de errores. Sin embargo, la verificación completa consume muchos recursos, de ahí que para sistemas de complejidad media o alta sea preciso recurrir a verificaciones parciales.
- El procesamiento de especificaciones formales puede automatizarse. Es posible construir herramientas software de ayuda al desarrollo, comprensión y depuración de especificaciones formales. Además una especificación formal puede ejecutarse (animarse) proporcionando así la posibilidad de un prototipo rápido.
- Las especificaciones formales pueden usarse para la definición de casos de pruebas.

1.9 El papel de los métodos formales en la Ingeniería del Software.

La fuerte base matemática (SOLLA 1999) de la mayoría de los métodos formales permite razonar sobre el sistema desde el primer momento, sin tener que esperar a la fase de codificación, donde los defectos de diseño están tan asentados que su corrección implica un gran coste económico.

Los beneficios potenciales que supone el poder razonar de forma temprana y constante serían:

- Una mejor comprensión del sistema.
- Una mejor comunicación con el cliente, al disponer de una descripción no ambigua de los requisitos de usuario.
- Una descripción más precisa del sistema (en cuanto a consistencia y compleción) y de las propiedades que debe cumplir.
- Una seguridad de carácter matemático de que el sistema implementado es correcto con respecto a su especificación.
- Una mayor calidad del software (entendiendo ésta como el grado de cumplimiento de las expectativas de los usuarios).

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, detectando posibles inconsistencias o ambigüedades que de otra forma podrían pasar sin ser vistas.

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 seguidores de los métodos formales son defensores de la idea de que el empleo de los mismos, a lo largo de todo el ciclo de vida del software, 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.

1.9 Pasos para aplicar los métodos formales.

Para la aplicación de los métodos formales existen un conjunto definidos de pasos para un correcto uso de los mismos.

1- EL primer paso es definir la invariante de datos, este se verifica mediante la ejecución de una función que contiene un conjunto de datos.

2- Definir el estado, que esta compuesto por datos a los que se puede acceder, y las operaciones, que van surgiendo a medida que se leen o escriben datos en el sistema, para el funcionamiento de un sistema

3- Definir la notación y la heurística asociadas con los conjuntos y especificaciones constructivas.

1.9 Panorámica de los Métodos Formales.

Actualmente, la aplicación de los Método Formales a la Ingeniería de Software es prácticamente inexistente fuera del ámbito académico, se limita a cierto tipo de sistemas informáticos de control en los que se debe contar con elevados niveles de confiabilidad y tolerancia a fallos, lo que no quiere decir que no se apliquen en proyectos. El desarrollo de herramientas que apoyen la aplicación de métodos formales es complicado y los programas que resultan son un poco incómodos para los usuarios. La mayoría de las personas en el mundo no conocen la existencia de los Métodos Formales por lo que la colaboración entre la industria y el mundo académico, es escasa.

Los métodos formales tienen una sólida base matemática por lo que mediante ellos se describen las propiedades de un software, además incluyen una notación precisa para la construcción de modelos matemáticos de un sistema. A través de ellos se encuentran las omisiones e inconsistencias en las declaraciones de requisitos más fiablemente que con otras técnicas. Los detractores aseguran que el empleo de métodos formales supone un volumen de trabajo considerable, aumento en los costes y tiempo de desarrollo y que debe quedar sujeto a herramientas que lo automaticen.

Sin embargo, el mundo industrial empieza a tomar otra actitud frente a los métodos formales, este cambio esta argumentado por varias razones por ejemplo: 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 las herramientas comerciales de calidad que soportan dichos métodos se consiguen en el mercado, pero estas no son todas las razones ya que a medida que los sistemas informáticos se hacen más complejos, las perdidas causadas por fallos en estos sistemas son cada vez más grandes.

En los últimos años la investigación en métodos formales crece en interés en el medio académico e industrial.

1.10 ¿Por qué utilizar métodos formales?

Los Métodos Formales mejoran en gran medida la calidad del producto permitiendo un mayor enfoque y entendimiento del problema, realizan una detección temprana de errores por lo que propician la creación de sistemas que sean más fiables y seguros, introducen rigor en el modelado y el análisis, permiten verificar la implementación y las herramientas asociadas simulan, animan, prueban, ejecutan y transforman el software.

Mirándolo desde el punto de vista de la productividad se evitan gastos y problemas habituales en los métodos tradicionales, pero se mantienen las ventajas competitivas de muchos procesos de desarrollo. En una línea distinta a la productividad, permiten evitar repercusiones legales como son la reducción del riesgo y la garantía de fiabilidad, corrección y seguridad, facilitando el cumplimiento de regulaciones y estándares que son de interés del gobierno en la seguridad de los sistemas y agencias de regulación y calidad.

La decisión de usar métodos formales (PRESSMAN 2006) debe considerar los costes de arranque, así como los cambios culturales asociados a una terminología totalmente diferente. En la mayoría de las instancias los métodos formales tienen mayores rendimientos respecto los sistemas cruciales de seguridad y los negocios.

Limitaciones de los métodos formales.

A pesar de ser muy útiles, los métodos formales no son del todo perfecto por lo que presentan algunas limitaciones a tener en cuenta:

- Desconocimiento del tema.
- Difícil de aplicar.
- La gran mayoría de profesionales, está adaptado a métodos no formales.

1.11 Aplicaciones generales.

Los métodos formales han sido aplicados al desarrollo del software de interfaces con el usuario con dos finalidades: para abstracción de los detalles de los usuarios, de su software, y de su interacción, la base para razonar y para analizar y asegurar una implementación correcta del software requerido.

Los Métodos Formales son apoyados por las aplicaciones donde el costo del fallo del software es mayor que el supuesto costo adicional del proceso del desarrollo formal. La experiencia industrial indica, de cualquier modo, que los costos adicionales (por ejemplo, la capacitación del personal que usará los métodos formales) son más que compensados por otros ahorros. Los métodos formales pueden ser usados para incrementar la confianza en la correctitud del software por demostración, refinamiento y pruebas.

Proyectos donde se han aplicado Métodos Formales.

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, participan investigadores de la Universidad Rutgers, de Estados Unidos y la Universidad de Victoria en Canadá, y aspira a combinar la Prueba Template y ClassBench para crear métodos y herramientas para realizar pruebas de software orientado a objetos.

Estos proyectos son utilizados para apoyar directamente la realización pruebas para el software de interfaz con el usuario. (HAYES I.J. 1990) Donde hacen uso de tres notaciones para especificar al mismo: los diagramas de transición de estado para expresar las secuencias de interacción, una notación basada en modelo similar a Z para definir las transiciones del estado y una notación algebraica para el razonamiento. Las pruebas son derivadas de los diagramas de transición de estado y las especificaciones son usadas para determinar la salida esperada.

El diseño de automatizaciones con PLCs está transformándose en la actualidad al uso de métodos formales que permite una calidad superior del sistema resultante, disminución del tiempo de diseño y de los gastos de implementación evitando posibles errores, así como de las exigencias del proceso a controlar, dadas principalmente por su seguridad, calidad, tiempo y costos de ejecución, ha provocado que actualmente la comunidad científica internacional muestre un creciente interés en el estudio y aplicación de métodos formales de diseño a este campo de la automática.

El proyecto CASENET (Computer-Aided solutions to Secure ElectroNic commerce Transactions) va a tomar como base el estado de la tecnología para la especificación, diseño, modelado y análisis formal de protocolos criptográficos para diseñar y desarrollar herramientas específicas que permitan a los usuarios, expertos o no, alcanzar los objetivos de seguridad que las aplicaciones a desarrollar han de contener, ya que con la utilización de los métodos formales para la especificación, diseño, modelado y análisis de sistemas este permite ayudar a construir sistemas más fiables.

La aplicación de los métodos formales a las arquitecturas basadas en agente es un tópico de investigación de actualidad. Así como el desarrollo del software orientado a objetos, las aproximaciones formales al desarrollo de las interfaces con el usuario basadas en agentes deberán dirigirse a ambos aspectos, el estático y el dinámico del sistema.

Se aplicó un modelo interactivo ubicuo aplicado al patrimonio natural y cultural del área del **Montsec**, la aplicación de este modelo a la zona del Montsec permitirá ofrecer un sistema itinerante de consulta y guía interactiva, con objeto de preservar y difundir los bienes naturales y culturales del parque. Como metodología de desarrollo se ha optado por complementar un método formal.

LIRA: Contorno software de desenvolvimiento de aplicaciones con técnicas de descripción formal, es una herramienta orientada al desarrollo de sistemas distribuidos con métodos formales. Esta aplicación ofrece soporte para el diseño formal de un sistema siguiendo el principio de refinamientos sucesivos. LIRA ofrecía ayuda al desarrollo transformacional de especificaciones LOTOS. En concreto, permitía definir transformaciones automáticas sobre una especificación LOTOS y verificar formalmente que se mantenían las propiedades de interés.

XLIRA es un proyecto financiado por la Xunta de Galicia. El resultado de este proyecto será la construcción de una herramienta software de soporte a todo el proceso de desarrollo propuesto

(la formalización del modelo constituye la base para la automatización), que incluye desde la fase de análisis y especificación de requisitos hasta las tareas de evolución y mantenimiento.

El trabajo del proyecto se centrará en la fase de especificación y análisis de requisitos (primera fase del modelo de desarrollo propuesto) cuyo objetivo es la obtención de la arquitectura inicial del sistema, sufrirá un conjunto de refinamientos cuyo objetivo es añadir el nivel de detalle suficiente para que la especificación pueda ser traducida a un lenguaje de implementación. La herramienta LIRA proporcionará el soporte adecuado para esta fase de refinamiento.

En la Universidad de Toronto se ha desarrollado un framework denominado i*, que permite obtener modelos organizacionales de empresas enfocándose en las metas que persigue cada proceso de negocios. En este marco de trabajo las organizaciones están constituidas por actores sociales los cuales tienen libertad de acción, pero dependen de otro actor para lograr sus metas, para desarrollar sus tareas y para que se le suministren recursos.

Se utilizan los métodos formales para incrementar la confianza en la correctitud del software de la interfaz con el usuario. 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.

Existe una asociación mundial que reúne investigadores de los métodos formales que desarrollan sistemas y software de cálculo. Los métodos formales se diferencian de muchas técnicas de la tecnología de dotación lógica en que tensionan la importancia de una base semántica rigurosa para las herramientas y las notaciones usadas. Tales fundaciones de los sonidos permiten el análisis de los diseños del sistema de cálculo a una profundidad que sería, de otra manera, imposible alcanzar. El objetivo de esta asociación es animar la investigación y el uso formales de los métodos formales.

La aplicación de los métodos formales a las arquitecturas basadas en agente es un tópico de investigación de actualidad. Así como el desarrollo del software orientado a objetos, las aproximaciones formales al desarrollo de las interfaces con el usuario basadas en agentes deberán dirigirse tanto al aspecto estático como al aspecto dinámico del sistema. Las notaciones basadas en modelo y basadas en propiedades son bien apropiadas para formalizar los aspectos estáticos, mientras que las notaciones basadas en comportamiento son preferibles para los aspectos dinámicos. Muchas aproximaciones formales basadas en agente utilizan notaciones múltiples de diferentes estilos, o extienden una notación existente para abarcar las capacidades de un estilo diferente.

Conclusiones

Se han estudiado los fundamentos teóricos relacionados con los métodos formales, donde se puede concluir que estos permiten mejorar la especificación informal dejándola sin ambigüedades por ende hacerlas más claras, completas y constantes las especificaciones; mediante la notación matemática y hacen que los errores se puedan descubrir rápidamente.

Con el concepto que se va a trabajar en los próximos capítulos es con el que da La Encyclopedia of software Engineering. Además el costo de utilizar métodos formales es muy elevado ya que se debe contar con un ingeniero experto en métodos formales, capacitar al personal en la notación formal y se debe contar con una persona que tenga grandes conocimientos matemáticos que le pueda explicar bien al cliente. Lo que no quiere decir que los ingenieros deben intimidarse con esto porque aplicarlos en los proyectos traería grandes beneficios para la calidad de los sistemas que se desarrollen.

CAPITULO 2: METODOS FORMALES PARA LA ESPECIFICACION DE REQUISITOS.

Introducción

En este capítulo se hace un estudio detallado de los diferentes Métodos Formales que se aplican a la especificación de requisitos para poder realizar una comparación crítica que permita valorar las ventajas de uno sobre otros, para determinar sus eficiencias, aunque se debe estar claro que no existe un método formal mejor que otro. Todos tienen sus puntos fuertes y debilidades y su elección depende del proyecto donde se vayan a utilizar.

2.1 Métodos Formales.

Los Métodos Formales tienen una base matemática, mediante ellos se describen las propiedades de un software, incluyen una notación precisa para la construcción de modelos matemáticos de un sistema (de software). Según Pressman a través de ellos se encuentran las omisiones e inconsistencias en las declaraciones de requisitos de manera más segura que con otras técnicas.

Se puede decir que los métodos formales son una parte importante del Proceso de software, debido a que permite verificar la calidad del diseño de software mediante evaluaciones comprobando si lo que se esta haciendo esta bueno o malo.

2.2 Métodos Formales basados en Modelos Matemáticos.

Los Métodos Formales basados en la Teoría de Conjunto y Lógica de Primer Orden son de verdadera importancia ya que las facilidades descriptivas que estas características ofrecen permiten un buen y claro planteamiento de los requisitos del sistema.

2.2.1 OCL (Object Constraint Language).

Desde los inicios de la informática se han estado utilizando distintas formas de representar los diseños de una forma más bien personal o con algún modelo gráfico. La falta de un estándar para graficar todo los modelos impedía entonces que lo diseños gráficos pudieran compartiese entre varios diseñadores. Para darle solución a esta dificultad y para servir de apoyo en los procesos de análisis de un sistema, es que surge UML (Unified Modeling Language).

UML es un lenguaje para especificar, visualizar, construir y documentar los artefactos de los sistemas software. Además es un método formal de modelado lo que brinda ventajas tales como una especificación con mayor rigor y la realización de una verificación y validación del modelo realizado. Un diagrama UML no está lo suficientemente refinado por lo que no refleja todos los aspectos notables de una especificación. Existe la necesidad de describir restricciones adicionales sobre los objetos en el modelo, esto siempre se traduce en ambigüedades, con el objetivo de resolver este problema surgen los Métodos Formales.

OCL (Object Constraint Language) es un componente de UML y a su ves, es una notación formal creada de manera tal que los usuarios de UML puedan realizar especificaciones con mayor precisión puesto que dispone de todo el poder de la Lógica y las matemáticas discretas. Sin embargo los diseñadores de OCL decidieron que en este lenguaje solo debería usarse caracteres ANSII en lugar de las notaciones matemáticas convencionales. Esto permite que el lenguaje sea más asequible a las personas menos inclinadas a las matemáticas y que las computadoras lo procesen con mayor facilidad.

Panorama de la sintaxis y la Semántica de OCL.

La utilización de OCL requiere que un ingeniero se software comience con unos o más diagramas UML como son diagramas de clases, estados o diagramas de actividad (estos son los mas comunes), por lo que se agregan expresiones OCL que establecen hechos acerca de los elementos de estos diagramas. Estas expresiones se llaman restricciones; cualquier implementación derivada del modelo debe garantizar que cada restricción siempre se mantenga

verdadera. Al igual que los lenguajes de programación, una expresión OCL involucra operadores que operan sobre objetos. Sin embargo, el resultado de una expresión completa siempre debe ser booleana.

El Método Formal OCL esta compuesto por:

Tipo de operandos	Operaciones
Real	=, +, -, *, /, abs, floor, max, min, <, >, <=, >=
Integer	=, +, -, *, /, abs, div, mod, max, min
Boolean	=, or, xor, and, not, implies, if-then-else
String	=, size, concat, toUpper, toLower, substring
Enumeration	=, <>

Tabla 1. Funciones aritmética y agregada.

Reglas de precedencia con un orden de precedencia de las operaciones, de mayor a menor prioridad como sigue:

- @pre
- punto y operaciones flecha: '.' y '->'
- operadores unitarios 'not' y menos '-'
- '*' y '/'
- '+' y el operador binario '-'
- 'if-then-else-endif'
- '<','>','<=','>='
- '=','<>'
- 'and','or', y 'xor'
- 'implies'

Los paréntesis '(' y ')' pueden usarse para variar la precedencia.

Operación collect: Se utiliza cuando se quiere especificar una colección que se deriva de otra, pero que contiene objetos diferentes de la colección original.

Operación ForAll: Se utiliza cuando se necesita que todos los elementos que componen la colección cumplan con una resticción determinada, esto especifica una expresión booleana que debe cumplir todos los objetos de una colección.

Operación Exists: Se utiliza cuando se necesita saber que hay al menos un elemento en una colección que cumple una restricción definida. Devuelve una expresión booleana.

En una expresión OCL, la palabra reservada **self** se usa para referirse a la instancia contextual de un objeto, esto es el elemento del diagrama UML en cuyo contexto se evaluara la expresión OCL. Se pueden obtener otros objetos al navegar usando el símbolo **.** (punto) del objeto self por ejemplo:(PRESSMAN 2006)

- Si **self** es clase **C** con atributo a entonces **self.a** evalúa el objeto almacenado en **a**.
- Si **C** tiene una asociación llamada **asoc** con otra clase **D**, entonces **self.asoc** evalúa un **Conjunto** cuyos elementos son del tipo **D**.
- Finalmente si **D** tiene un atributo **b**, entonces la expresión **self.asoc.b** evalúa el conjunto de todas las b que pertenecen a todas las **D**.

OCL también se utiliza para especificar pre-condiciones y pos-condiciones de operaciones, tiene las características de un lenguaje de expresiones, un lenguaje de modelos y un lenguaje formal:

Un lenguaje de expresión puro: OCL es una notación de expresiones pura, esto implica que no puede cambiar nada en el modelo, significa que un estado del sistema nunca cambiará debido a una expresión OCL, incluso una expresión OCL podría usarse para describir un cambio de estado. Todos los valores de todos los objetos, no cambiarán. En cualquier momento en que se evalúa una expresión OCL, simplemente devuelve un valor.

Un lenguaje de modelado: OCL no es un lenguaje de programación lo que implica que no se

puede escribir un programa lógico o un flujo de control en OCL. Especialmente, no se puede invocar procesos o activar operaciones de consulta del mismo debido a que OCL es en primer lugar un lenguaje de modelos, no se puede asegurar que todo sea directamente ejecutable.

Un lenguaje formal: OCL es un lenguaje formal donde todos los constructores tienen un significado formal definido. La especificación de OCL es parte de la especificación de UML. OCL no tiene la intención de reemplazar los lenguajes formales existentes. Los lenguajes formales tradicionales se usados por personas con conocimientos matemáticos, pero dificulta su uso para la mitad de empresas y modeladores de sistemas por lo que se desarrolla OCL con el objetivo de fortalecer este aspecto, puesto que en un proyecto hay mucha gente involucrada (usuario, expertos, otras personas) y los modelos deben ser entendidos por todas ellas.

Fácil de usar por el personal de desarrollo: Los lenguajes formales tradicionales son difíciles de utilizar por personas que no tengan una buena base en matemáticas pero OCL ha sido pensado para quien típicamente se dedica al modelado de negocios o sistemas.

OCL es fácil de aprender y usar por los desarrolladores sin amplios conocimientos matemáticos, tiene muchas características que posibilitan a los mismos acogerlo a su ritmo y solo donde lo necesitan. OCL es un lenguaje tipado, por lo que cada expresión OCL tiene un tipo. Para ser bien formada, una expresión debe concordar con los tipos de las reglas del lenguaje. Por ejemplo, no puede compararse un Integer con un String. Cada clasificador definido en un modelo UML representa un tipo distinto en OCL. Además, OCL incluye un conjunto de tipos adicionales predefinidos.

Utilizando OCL se tiene la posibilidad de escribir las especificaciones completas, ya que la intención de la creación de este método formal es la de utilizarlo en combinación con los modelos visuales UML. Muchos aspectos del modelado pueden expresarse mejor usando diagramas y OCL no intenta reemplazarlos por sus propios mecanismos, sino que toma esa información visual y permite al desarrollador acceder a esta información en las expresiones OCL.

Le Lenguaje OCL tiene diversas utilidades, todas con propósitos diferentes dada la gama de características con la que cuenta:

- Para especificar invariantes sobre clases y tipos en el modelo de clases.
- Para especificar pre y post-condiciones sobre operaciones y métodos.
- Para describir guardas.
- Como lenguaje de navegación.
- Para especificar restricciones sobre operaciones.

Utilización de un paquete estándar OCL.

Todo modelo UML que usa como lenguaje de restricción OCL contiene un paquete estándar denominado “UML_OCL”, este contiene todos los tipos OCL predefinidos y sus características además de ser usado por defecto en todos los otros paquetes en el modelo para evaluar expresiones OCL.

Para extender los tipos predefinidos OCL, un modelador debe definir un paquete separado. El paquete estándar de OCL puede ser importado, y cada tipo puede extenderse con nuevas características. Para especificar que un paquete usó tipos predefinidos OCL de un paquete definido por el usuario en lugar de un paquete estándar, el paquete usado debe definir una dependencia con estereotipo <<Tipo_OCL>> al paquete que define el tipo OCL extendido.

El paquete de usuario debe ser una extensión correcta del estándar OCL. Este lenguaje no es tan expresivo como el cálculo relacional, está incompleto como lenguaje de consulta en el sentido de bases de datos. Hace práctica la construcción de modelos más formales y de esta forma es más fácil encontrar los errores en fases más tempranas del proyecto. La combinación de UML y OCL

mejora definitivamente el proceso de desarrollo software e incrementa la calidad del software desarrollado.

2.2.2 Z (Zeta)

El Lenguaje de especificación formal Z o la Notación Z o simplemente Z como también se conoce, es un lenguaje matemático, comenzó a desarrollarse en la Universidad de Oxford, en el año 1980. Su nombre deriva del matemático Zermelo.

Z es un lenguaje de especificación basado en estados los cuales representa mediante los datos almacenados y los define como operaciones que ocurren dentro del sistema que lee o escribe datos. Es un sistema caracterizado por un conjunto de estados entre los que existe un estado inicial y un conjunto de operaciones que definen las transiciones de un estado a otro. Este método es especializado en la especificación del comportamiento de sistemas secuenciales.

La notación Z es utilizada para modelar una gran variedad de sistemas secuenciales ya que sustenta su base teórica en:

- La teoría de conjuntos cuyos conceptos permiten la modelación de estructuras de datos.
- La lógica de predicados de primer orden que se usa para describir de forma abstracta el efecto de cada operación en el sistema.
- Las teorías de relaciones y funciones.

Aunque los conceptos de estas teorías son la base de Z, no se usará la notación clásica en ellas. Z como lenguaje de especificación hace uso de símbolos gráficos y cajas que, habitualmente no se encuentran disponibles en un sistema editor estándar.

Además existe un lenguaje basado en Z que se denomina Z++ (SPIVEY 1992) el cual fue propuesto como lenguaje de diseño, además de poseer elementos para estructurar incrementalmente desarrollos a gran escala. Sintácticamente, una clase consta de nombre, parámetros opcionales de tipos y un conjunto de condiciones que describen características como: extensión desde otras clases previamente definidas, definición de tipos locales a una clase, variables locales representando atributos al estilo Z, operaciones sin efectos laterales, invariante sobre el estado interno, operaciones con efectos laterales (acciones) que se pueden especificar como predicados Z.

Las acciones pueden cualificarse como espontáneas y un predicado historia describirá las ejecuciones admisibles para los objetos de la clase. En vez de utilizar la notación de cajas de Z se utiliza una notación ASCII similar a la de un lenguaje de programación orientado a objeto. Se admite la parametrización de tipo, sin embargo debe ser instanciada una clase genérica para ser considerada como tipo, ya que ella por sí misma no lo es.

En este lenguaje se admiten métodos de clase ya que Z++ permite varias operaciones y relaciones sobre las mismas, hay algunas operaciones para ocultar y renombrar características de una clase, instanciar clases genéricas, unión, conjunción, refinamiento, equivalencia y herencia entre clases.

Panorama de la Sintaxis y la Semántica de Z.

Las especificaciones Z están organizadas como una estructura que se le parece a un recuadro que introduce variables y las relaciones entre ellas, a esto le llamamos esquema que es la especificación formal análoga a un componente de un lenguaje de programación. Estos describen los datos almacenados a los que el sistema accede y modifica. Además el esquema identifica las operaciones que se aplican para cambiar estados y las relaciones que ocurren dentro del sistema.

Estructura genérica de un esquema:

_____ nombreEsquema_____

Declaraciones

_____

Invariante

Donde las declaraciones son las encargadas de identificar las variables que comprenden el estado del sistema y el invariante impone las restricciones en la forma en que el estado puede evolucionar. Por convención en la notación Z una variable de entrada que no forma parte del estado y se lee termina en un signo de interrogación.

Conjuntos:	
$S : \mathbb{N}\ X$	S se declara como un conjunto de *X*.
$x \in S$	x es miembro de *S*.
$x \notin S$	x no es miembro de S
$S \subseteq T$	S es un subconjunto de *T*: Todo miembro de S está también en *T*.
$S \cup T$	La unión de S y *T*: Contiene todos los miembros de S o *T* o ambos.
$S \cap T$	La inserción de S y *T*: Contiene todos los miembros tanto de S como de *T*.
$S \setminus T$	La diferencia de S y *T*: Contiene todos los miembros de S salvo los que están también en *T*.
$\emptyset$	Conjunto vacío: No contiene miembros.
{x}	Conjunto unitario: Solamente contiene a x.
$\mathbb{N}$	El conjunto de los números naturales 0, 1, 2....
$S : \mathbb{F}\ X$	Se declara S como un conjunto finito de *X*.
max (*S*)	El máximo del conjunto no vacío de números *S*.

Funciones: $f: X \rightarrowtail Y$ dom f ran f $f \oplus \{x \mapsto y\}$ $\{x\} \lhd f$	Se declara como una inyección parcial de X e Y. El dominio de f: Dícese del conjunto de valores de x para los cuales está definido $f(x$ El rango de f: El conjunto de valores que toma $f(x)$ cuando x recorre el dominio de f. Una función que coincide con f salvo que x se hace corresponder con y. Una función igual que f, salvo que x se ha eliminado de su dominio.
Lógica: P A Q $P \Rightarrow Q$ $\theta S' = \Theta S$	P y Q: Es verdadero si tanto P como Q son verdaderos. P implica Q: Es verdadero tanto si Q es verdadero como si P es fals Ningún componente del esquema S cambia en una operación.

Tabla 2. n abreviado conjunto de símbolos de Z

Proyectos donde se ha aplicado Z.

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 de una orden de magnitud mayor que las correspondientes de los promedios de la industria para la métrica, líneas de código por persona al mes. Un 36% mejores que los promedios de la industria.

Actualmente un proyecto en la Universidad de Queensland esta interesado en las notaciones formales para las especificaciones (orientadas a objeto) del software de la interfaz con el usuario y el desarrollo de diseños utilizando transformaciones relativamente informales basadas en patrones de software, una extensión de Z orientada a objetos, para las especificaciones de interactores. El Object-Z enriquece las capacidades de Z basadas en modelo y con encapsulación construye e incluye operadores para expresar aspectos dinámicos tales como la concurrencia y la comunicación entre objetos.

El segundo modelo interactor AMODEUS fue desarrollado en la Universidad de York, Reino Unido. El modelo de York utiliza notaciones basadas en modelo como Z y VMD, aumentadas por otro método para aspectos de comportamiento.

CICS (Customer Information Control System) es un proyecto desarrollado por IBM en colaboración con el Laboratorio de Computación de la Universidad de Oxford. En 1992 se utilizó Z para la nueva versión del sistema de procesamiento de transacciones de la versión CICS/ESA V3.1.

CIDS: Z se utilizó como especificación formal de parte del software del Airbus A330/340.
Estas aplicaciones es algo que de lo que se ha hecho en el mundo con Z que es un de los métodos más usados.

Ventajas y Limitaciones de Z.

La notación Z es un lenguaje de especificación muy expresivo para describir las entidades de los sistemas y aplicaciones software, aunque también presenta otras limitaciones, se plantea que la Notación Z no es adecuada para especificar sistemas reactivos. Efectivamente, Z es muy apropiado para modelar las entidades estáticas y su comportamiento, pero no para modelar conceptos dinámicos como puede ser el tiempo real o la concurrencia. Sin embargo se ha probado que es posible vencer algunas de estas limitaciones, introduciendo genericidad (para especificar comportamiento concurrente en Z), tiempo real, modularidad (para encapsular componentes concurrentes dentro de una especificación Z) y comunicación síncrona. Algunas de estas son realmente elegantes y simples, mientras que otras (como la modularidad) se consiguen quizás de una forma ligeramente forzada y antinatural.

También puede verse ligado a la falta de modularidad de Z otra de sus limitaciones: la falta de estructura en las especificaciones, lo que complica mucho aquellas de gran tamaño ya que este

método opera con variables globales, y no es posible realizar especificaciones usando una estructura que no sea plana. Por otro lado, una de las grandes ventajas de Z es que ofrece una metodología y un proceso para la derivación de implementaciones a partir de las especificaciones.

Por la necesidad de introducir la programación orientada a objeto en Z se manejaron las idea de mantener a Z siguiendo un estilo Orientado a Objetos o extender Z por sus características favorables para construir notaciones realmente orientadas a objetos es decir que proporcionaran un mayor alcance es así que surge Object-Z.

Haciendo énfasis en Object-Z se puede decir que es una extensión de Z (SPIVEY 1992) y que a diferencia de Z en Object-Z, una operación se refiere sólo al estado del objeto al que pertenece y cada clase se obtiene mediante la definición de estado y operaciones que tengan asociadas.

Object-Z trata de solucionar uno de los inconvenientes de Z, permitiendo agrupar un estado con un conjunto de operaciones sobre ese estado, en el más puro estilo de orientación a objetos, para ello define el concepto de clase, permite declarar objetos de esa clase, así como extender las clases mediante herencia, incorporando genericidad a sus clases y la posibilidad de incluir invariantes de clase.

Herramientas proporcionadas para Z.

CADiZ, DST Z-Toolbox, ForMooz, PiZA, VisualiZer, Z Browser, Zola, ZANS [Xia95] y Cogito [ABT95].

2.2.2 VDM (Vienna Development Method)

VDM (Vienna Development Method) es una colección de técnicas para la especificación formal y el desarrollo de sistemas software. Tiene su origen en un laboratorio de IBM entre 1960 y 1970. Este método es especializado en la especificación del comportamiento de sistemas secuenciales. El modelo del sistema se detalla mediante un conjunto de tipos de datos y una serie de operaciones sobre los mismos que se representan mediante funciones. A cada función se le asocia una pre-condición y una post-condición, donde las pre-condiciones definen circunstancias en las cuales es valida una operación particular y las pos-condiciones definen lo que esta garantizado que será cierto hasta completar la condición.

Utiliza un lenguaje de especificación denominado VDM-SL, mediante este se definen tipos complejos como conjuntos, registros, secuencias, etc., sobre los que se pueden especificar restricciones mediante invariantes y operaciones. Resulta ser menos potente que Z en la etapa de análisis pero más adecuado para la etapa de diseño.

También utiliza otro lenguaje de especificación basado en VDM-SL (JONES 1990) que se le llama VDM++, la cual consta de un conjunto de definiciones de clases. Una definición de clase representa una plantilla para objetos con atributos, operaciones (métodos) y propiedades sobre concurrencia y tiempo real, además las clases consta de varias condiciones para definir constantes, variables, invariantes, métodos, tareas y sincronizaciones. Los tipos, constantes y métodos se heredan de VDM-SL.

Un invariante restringe los valores iniciales de las variables y los valores antes y después de la ejecución de un método. Una clase puede ser declarada como subclase de otra por lo que toda característica de la clase se convierte en característica de la subclase. Adicionalmente, se pueden añadir nuevos atributos, invariantes, métodos, etc.

Proyectos donde se ha aplicado VDM.

Dos de los proyectos más relevantes en los que se ha utilizado VDM son:

CDIS Parte del mecanismo de representación de información del sistema de control de tráfico aéreo de Londres fue verificado utilizando VDM. Este proyecto fue llevado a cabo en 1992 por la empresa Praxis.
ACS: En este caso, VDM se utilizó para especificar requisitos de seguridad de un sistema de almacenamiento de explosivos del ejército británico.

Por lo antes expuesto se puede ver que este método es preciso y verificable, con una sintaxis y una semántica bien definidas pero a su vez difíciles de entender comunicar a los clientes debido a la complejidad de sus propuestas.

Herramientas que utilizan VDM.
IFAD VDML-SL Toolbox, Mural, SpecBox.

2.3 Técnicas basadas en Algebra de Procesos.

Las técnicas basadas en Algebra de Procesos, son lenguajes especializados en la descripción de sistemas concurrentes, la concurrencia se expresa mediante procesos secuenciales que interactúan en paralelo, comunicándose y sincronizándose mediante canales.

Cada proceso se especifica mediante una notación en la que las variables definen los estados, y las constantes o símbolos terminales especifican las acciones, eventos y transiciones. Los sistemas se construyen mediante conjuntos de procesos, que se componen utilizando distintos operadores que modelan la elección, ejecución paralela y comunicación entre los procesos. Estos operadores constituyen un álgebra de procesos.

2.3.1 Las Máquinas de Estados Finitos.

Los Lenguajes formales que se consideran Métodos Formales cuentan con esquemas reconocedores del lenguaje. Se entiende por reconocedor a un programa que permite decir si la cadena pertenece o no pertenece al lenguaje, teniendo como esquemas reconocedores del lenguaje los autómatas finitos que son casos especiales de reconocedores, ya que permite construir la fase de análisis, reconocer un determinado lenguaje regular y permite reconocer si una cadena recibida pertenece al lenguaje que el autómata reconoce.

Las Máquinas de Estados Finitos son autómatas finitos con salida por lo que también se les conoce como Autómatas de Estados Finitos, se explica, son modelos de comportamiento de un sistema o un objeto complejo, con un número limitado de modos o condiciones predefinidos, donde existen transiciones de modo.

Están compuestas por 4 elementos principales:

- Estados que definen el comportamiento y pueden producir acciones.
- Transiciones de estado que son movimientos de un estado a otro.
- Reglas o condiciones que deben cumplirse para permitir un cambio de estado.
- Eventos de entrada que son externos o generados internamente, que permiten el lanzamiento de las reglas y permiten las transiciones.

Una Máquina de Estados Finitos debe tener un estado inicial que actúa de punto de comienzo, y un estado actual que recuerda el producto de la anterior transición de estado. Los eventos recibidos como entrada actúan como disparadores, que causan una evaluación de las reglas que gobiernan las transiciones del estado actual a otro estado. La mejor manera de visualizarla es pensar en ella como un diagrama de flujo o un grafo dirigido de estado, aunque existen técnicas de abstracción más precisas que pueden ser usadas. Las Máquinas de Estado Finitos se usan típicamente como un tipo de sistema de control donde el conocimiento está representado en los

estados, y las acciones están restringidas por las reglas. Son una técnica adoptada por la inteligencia artificial que se originó en el campo de las matemáticas, inicialmente se utilizó para la representación de lenguajes.

Una Máquina de Moore es un tipo de máquina de estados finitos donde las salidas se generan como producto de los estados. En el ejemplo de abajo los estados definen que hacer; como por ejemplo encender la bombilla. Normalmente a la especificación de requisitos se le hace una verificación ya que no están exentas de errores o ambigüedades, las cuales se realizan a través de reuniones e inspecciones de revisión. No obstante la utilización de métodos formales brinda ventaja respecto a este punto ya que la base matemática que ellos poseen posibilita la verificación formal de propiedades formuladas sobre la especificación y verificación formal de que la implementación satisface la especificación.

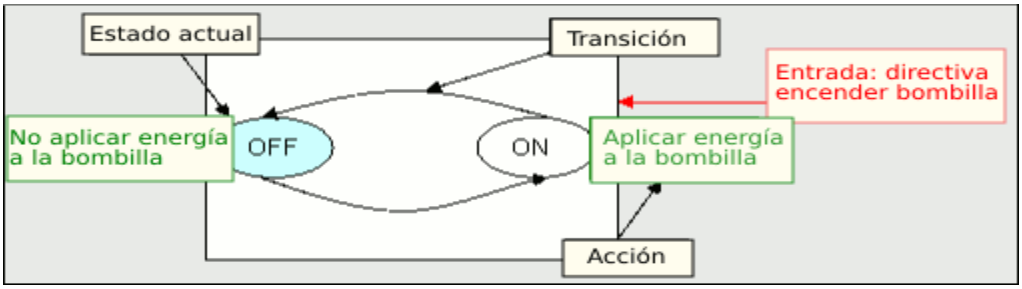

Figura 1. Sistema de luz ejemplo de una Máquina de Moore

Una Máquina de Mearly, a diferencia de la Máquina de Moore es una máquina de estados finitos donde las salidas se generan como producto de las transiciones entre estados. En el ejemplo de abajo la luz se ve afectada por el cambio de estado.

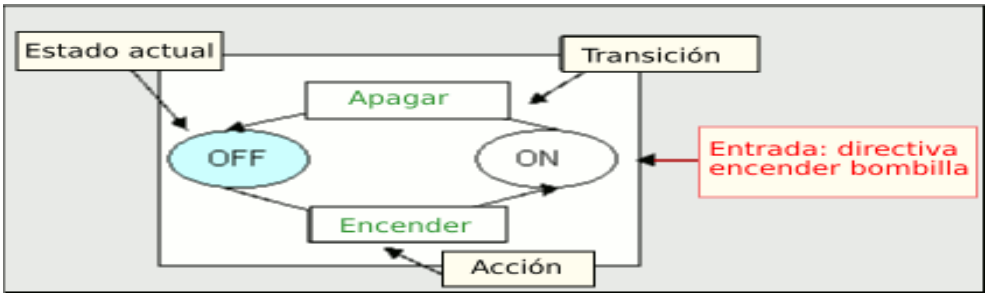

Figura 2. Sistema de luz ejemplo de una Máquina de Mearly

Las máquinas de estados finitos no son una nueva técnica, ya que hace tiempo que existen. El concepto de descomposición debe ser familiar a las personas con experiencia de programación o diseño. Existen varias técnicas de modelado abstracto que pueden ayudar o mejorar el entendimiento al definir y diseñar una máquina de estados finitos, la mayoría provienen del área del diseño o las matemáticas.

Ventajas de una Máquina de Estado Finito.

- Su simplicidad hace fácil a los desarrolladores sin experiencia realizar la implementación con poco o nada de conocimiento extra (fácil entrada).
- Dado un grupo de entradas y un estado actual conocido, puede predecirse la transición de estados, facilitando la tarea de verificación.
- Dada su simplicidad, las máquinas de estados finitos son rápidas de diseñar, rápidas de implementar y rápidas de ejecutar
- Las Máquinas de Estados Finitos son relativamente flexibles.
- La transferencia desde una representación abstracta del conocimiento a una implementación es fácil.
- Es fácil determinar si se puede llegar o no a un estado y que requerimientos existen para hacerlo.

Desventajas de una Máquina de Estados Finitos.

- La naturaleza predecible de las Máquinas de Estados Finitos deterministas puede no resultar conveniente en algunos dominios como los juegos por ordenador.
- Si se implementa un sistema grande usando Máquinas de Estados Finitos puede ser difícil de administrar y mantener sin un buen diseño.
- No es apropiado para todos los dominios de problema, solo debe ser usado cuando el comportamiento de un sistema, puede ser descompuesto en estados separados con condiciones bien definidas para las transiciones. Esto significa que todos los estados, transiciones y condiciones deben ser conocidos y estar bien definidos.
- Las condiciones para las transiciones entre estados son rígidas, significando que están fijadas.
- Las máquinas de estados finitos son subjetivas y dependen de cada problema específicamente.
- Está claro que las Máquinas de Estados Finitos están bien adaptadas a dominios de problemas que se expresan fácilmente usando diagramas de flujo y poseen un grupo de estados y reglas que gobiernan las transiciones entre estados bien definidos.

2.3.2 CSP (Calculus of Sequential Processes).

CSP (Calculus of Sequential Processes) fue definido en 1978 se le realizan mejoras en 1985 por lo que se convierte en una notación más flexible. Se crea para describir concurrencia y comunicación entre procesos. CSP permite describir un sistema a través de un conjunto de componentes (procesos) que operan independientemente y que se comunican entre ellos por canales definidos para tal propósito, el nombre se mantiene aunque la restricción de su aplicación

a sistemas secuenciales se eliminó entre 1978 y 1985. El modelo de comunicación de CSP, es en el que la información se transmite solamente a través de canales nombrados explícitamente.

Proyectos donde se ha aplicado CSP.

Schneider S. en 1998 en su trabajo Verifying authentication protocols in CSP, utilizó CSP para modelar protocolos en un entorno hostil y para expresar propiedades de seguridad. La verificación se llevaba a cabo descubriendo una función de clasificación.

Ventajas y limitaciones de CSP.

CSP es un método formal que facilita la comunicación ya que encapsula sincronización, planificación y transferencia de datos. Además su ausencia de buffer, proporciona un buen mecanismo de sincronización entre procesos, por lo que la implementación en multicomputadoras es muy eficiente.

Pero a pesar de ser creado con todas estas facilidades, a CSP se le presentan en varias ocasiones un problema de satisfacción de restricciones, es decir, que puede evaluar todos los posibles valores de asignación a las variables del problema, buscando aquella solución que cumpla con todas las restricciones del problema

Herramientas.

FDR, ProBE, Caspe.

2.3.3 SDL (Specification and Description Languange).

SDL (Specification and Description Languange) es un lenguaje de descripción formal normalizado por ITU (International Telecommunication Union). Desde su surgimiento, SDL ha sufrido

constantes revisiones. Las extensiones más recientes del lenguaje se han realizado en el área de diseño orientado a objetos.

En este Método Formal 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 que favorece el desarrollo por refinamientos sucesivos.

El sistema se describe como un conjunto de máquinas de estados finitos y extendidos, comunicándose mediante el intercambio de mensajes entre ellas y con el entorno esto lo convierte en un lenguaje muy apropiado para representar sistemas basados en estímulo-respuesta.

Proyectos donde se ha aplicado SDL.

Es un lenguaje de amplia utilidad, tanto en su utilidad desde la fase de especificación requisitos hasta la de implementación como en su ámbito de aplicación en sistemas de telecomunicaciones, control ferroviario, aplicaciones médicas.

Dos de los proyectos más relevantes en los que se utilizó la técnica formal SDL son:

- INSYDE: Proyecto Sprit dedicado al diseño híbrido hardware-software. SDL se utilizó para la especificación del software de un sistema de vídeo bajo demanda.
- NewCore: Es un sistema de telecomunicaciones en el que SDL se utilizó para su verificación.

Herramientas que utilizan la técnica formal SDL.

SDL es, probablemente, la técnica formal más utilizada en la Industria existiendo múltiples herramientas, tanto comercial como de dominio público donde se destacan:

- QUEST: Proporciona un entorno para la descripción e integración de subsistemas, el análisis de requisitos, la visualización de comportamiento y el estudio de rendimientos.
- ProcGen: Es una herramienta comercial de desarrollo de sistemas por refinamientos sucesivos que permite generar código de forma automática.

2.3.4 ESTELLE (Extended State Transition Language).

ESTELLE (Extended State Transition Language) fue desarrollado inicialmente para especificar sistemas distribuidos de información o concurrentes y luego lo concibieron, en particular, para desarrollar protocolos de comunicaciones bajo el modelo de referencia OSI, forma parte de una segunda generación de Técnicas de Descripción Formal.

Esta técnica formal de especificación esta basada en Máquinas de Estados Finitos y extendidas, normalizada por ISO en 1989 y 1991, por lo que un sistema se modela como una jerarquía de autómatas por lo que pueden ejecutarse en paralelo y pueden comunicarse entre sí intercambiándose mensajes o bien compartiendo variables de forma restringida. Tiene su fundamento sintáctico es el Pascal ya que las acciones que se realizan en los distintos módulos o en las transiciones se especifican con este lenguaje y consta de una fuerte estructura de objetos tipados, lo que permite detectar las errores de la especificación en la fase de compilación.

Ventajas de ESTELLE.

La principal ventaja de la utilización de ESTELLE es que las especificaciones generadas de los

sistemas descritos mediante Máquinas de Estados Finitos, sin dejar de ser formales, se acercan mucho a la implementación final, por lo que suelen generarse códigos muy eficientes a partir de ellas. Además permite definir sistemas jerarquizados y con diferentes tipos de paralelismos, característica necesaria para el desarrollo de sistemas descritos mediante un conjunto de Máquinas de Estados Finitos.

En ESTELLE las descripciones se basan en conceptos muy familiares ya que las especificaciones de los sistemas son muy naturales, además proporcionan al equipo de implementación del sistema una guía de trabajo.

Herramientas.

Herramientas que traducen automáticamente ESTELLE a C++ como son:
PetDingo, XEC y a C es **EDT** y que incluyen simuladores y ayudas a la depuración.

2.3.5 LOTOS (Language of Temporal Ordering Specification).

LOTOS (Language Of Temporal Ordering Specification) es una técnica de descripción formal normalizada en 1989 por la norma ISO, un lenguaje formal de especificación basado en el álgebra de procesos de un sistema, permitiendo realizar, mediante un análisis técnico, la verificación, validación y desarrollo de dicho sistema.

Su principal objetivo es permitir la especificación formal de los sistemas distribuidos, expresar formalmente normas de servicios y protocolos OSI de una manera clara, no ambigua, precisa, completa, e independiente de la implementación por lo que en una especificación LOTOS, un sistema se modela como una colección de procesos que se componen secuencialmente y en paralelo para definir el comportamiento observable del sistema que no es más que el conjunto de todas las posibles secuencias de interacción en las que puede participar, por lo que en este método un sistema concurrente y distribuido, es visto como un proceso, consistente de varios subprocesos, por lo que describe un sistema mediante la definición jerárquica de procesos.

Un proceso es una entidad capaz de realizar acciones internas, no observables, así como interactuar con otros procesos, las relaciones complejas entre los procesos se construyen a partir de los eventos los cuales sirven para la sincronización de los procesos. Dicha sincronización puede incluir el intercambio de datos.

Operadores de LOTOS.

La sintaxis de LOTOS esta conformada por un aserie de operadores:

Prefijo (**;**):

Selección (**[]**)

Paralelismos (**|||**)

Sincronización (**||**)

Compuertas (**|[]|**): Este operador trabaja con listas y es el que cierra el conjunto básico del álgebra de procesos.

Terminación (**exit** y **stop**): El stop significa un deadlock o inactividad y exit significa salida exitosa del proceso.

Habilitar Procesos (**>>**): Es una generalización del operador de prefijo y se usa para combinar dos expresiones en una secuencia, implica que si el proceso de la izquierda termina con el operador stop pues otro nunca se ejecutara.

Deshabilitar procesos (**[>**): Se utiliza cuando se hace necesario interrumpir procesos si se cumplen ciertas condiciones

No determinismo (**CHINESE_ MEAL** o **INDIAN_ MEAL**): Debido a una selección, cuando dos procesos ofrecen el mismo evento.

LOTOS tiene dos partes claramente diferenciadas. La primera parte facilita un modelo basado en álgebra de procesos para expresar el comportamiento e interacción entre ellos. La segunda parte del lenguaje permite la especificación de tipos abstractos de datos. En la actualidad, LOTOS se encuentra en proceso de revisión. Los trabajos de revisión, que se encuentran en una fase avanzada, darían lugar a una nueva versión de LOTOS denominada E-LOTOS la cual se encuentra todavía en proceso de estandarización. Esta versión es principalmente promovida en Europa y pretende superar algunas de las carencias del lenguaje original.

Proyectos en lo que se ha utilizado LOTOS.

En proyectos dedicados a la automatización como MEDAS, EMEDAS y OTELSO se ha trabajado en la integración de los métodos formales en procesos de desarrollo Software industrial. En este sentido se han desarrollado reglas para regir su metodología, entornos de herramientas y casos de estudio para la aplicación de Métodos Formales como LOTOS (Language of Temporal Ordering Specification) y SDL (Specification and Description Languange) en el desarrollo de aplicaciones distribuidas de telecomunicación.

Herramientas.

Además LOTOS cuenta con herramientas para construir expresiones lógicas y su sintaxis es compatible con ProLog, con esto se sabe que su campo de acción es bastante potente, herramientas como LOLA, ISLA, CADP y NOTOS.

2.3.6 Lógica Temporal.

La lógica temporal es una disciplina matemática que permite razonar sobre el transcurso del tiempo, ya sea de forma cuantitativa o cualitativa, este último aspecto permite establecer restricciones sobre la ordenación que ciertos eventos que pueden o deben tener a lo largo de un período de tiempo. Mediante este método formal se realiza la especificación del comportamiento de forma implícita, al restringir las ordenaciones temporales posibles de sus eventos. La

aplicación de la lógica temporal a la especificación y verificación de sistemas software ha sido de gran utilidad en el proceso de razonamiento sobre programas.

La lógica temporal en general es un formalismo muy adecuado para el estudio de sistemas reactivos (PNUELI 1985) que son aquellos cuyo comportamiento no consiste en una serie de pasos ordenados que finalizan produciendo un resultado, sino que se perpetúa en una constante interacción con su entorno, sin un fin previsible. Por lo que se considera que las distintas versiones de la lógica temporal, con distinto grado de adecuación, permite:

Describir la evolución de un sistema a través de la ordenación temporal de los eventos que determinan su interacción con el entorno. Mediante la lógica temporal se realiza la especificación del comportamiento de forma implícita, al restringir las ordenaciones temporales posibles de sus Eventos.
Describir ciertas propiedades cuya verificación en un sistema se considera interesante. Esto se suele realizar, también, mediante restricciones sobre los ordenamientos posibles. Razonar sobre el cumplimiento de una propiedad en un sistema.

Ventajas de la Lógica Temporal.

La principal ventaja de la lógica temporal es su capacidad para tratar con éxito el comportamiento antes y después de los programas. Además, otra de las ventajas más importantes de la lógica temporal es que nos permite trabajar de forma directa con especificaciones parciales. La relación de satisfacción que se define en el campo de la lógica nos permite proceder a la verificación de propiedades en una parte del sistema, sin necesidad de tratar con la especificación completa.

Limitaciones de la Lógica Temporal.

Su principal inconveniente reside en su incapacidad para reflejar el comportamiento progresivo de los programas: la variable x tomaría el valor 0 en algún momento de la ejecución del sistema. Se han desarrollado algunas lógicas de procesos para enmendar esta dificultad.

Herramientas

Concurrency Workbench, JACK, y ATG FC-TOOLS.

2.3.7 TRIO (TRIO+).

TRIO es un lenguaje formal basado en la lógica temporal de primer orden, ideado para construir especificaciones ejecutables de sistemas en tiempo real, ya que este modela el mismo en forma cuantitativa. TRIO proporciona soporte para diferentes actividades de validación como testar especificaciones, simulación y pruebas. En (MORZENT 1994) se proponen la ejecutabilidad de las especificaciones como medio de asegurar la adecuación de los requisitos.

TRIO+ se diseñó para construir especificaciones de sistemas complejos de forma sistemática y modular, consta de un conjunto de definiciones de clases donde cada una de ellas es un conjunto de axiomas, estas clases pueden ser simples o estructuradas. Donde las clases simples contienen una interfaz (métodos), un dominio temporal (real racional, entero), declaración de predicados dependientes del tiempo, variables, funciones, colección de axiomas. Las clases complejas son clases con módulos por lo que la semántica de una clase compleja se establece mediante clases simples equivalentes sin los mecanismos de estructuración utilizados. Es importante destacar la utilización de TRIO+ en la industria ENEL Italian Electric and Energy Board.

2.3.8 Redes de Petri.

Las Redes de Petri fueron inventadas por el alemán Karl Adam Petri en 1962, las cuales representan una opción para modelar el comportamiento y la estructura de sistemas sus

características hacen que funcionen de una manera natural. Una red de Petri permite manipular eventos que son activados bajo condiciones explícitas.

Una red de Petri es un grafo dirigido orientado con dos tipos de nodos: lugares (representados mediante circunferencias) y transiciones (representadas por segmentos rectos verticales), de tal forma que los arcos van de un lugar, que seria una entrada, a una transición que seria entonces el lugar de entrada o de una transición a un lugar que sería lugar de salida.

En su representación gráfica, los lugares se suelen dibujar como círculos y puede contener cero o más marcas, y las transiciones como barras. Los arcos vienen etiquetados con pesos (enteros positivos), donde un arco de peso k puede interpretarse como un conjunto de k arcos paralelos. Un marcado (o, lo que es lo mismo, un estado) asigna a cada lugar un entero no negativo.

Estas redes han sido ampliamente utilizadas en el modelado de sistemas cuyo comportamiento es guiado por un patrón de reglas complejas.

Estructura de una red de Petri.

Las Redes de Petri se componen de cuatro partes:

- Un conjunto de nodos.
- Un conjunto de transiciones.
- Una función de entrada y
- Una función de salida.

Las funciones de entrada y salida relacionan a los nodos y a las transiciones. La función de entrada es un mapeo de una transición t_j a una colección de nodos conocidos como los nodos de entrada de una transición. La estructura de una PN es definida por los nodos, las transiciones, la función de entrada y la función de salida.

Modelo de Redes de Petri.

El modelo de redes de Petri es suficientemente general como para representar una gran variedad de sistemas y es especialmente idóneo para todos aquellos que tienen un comportamiento asíncrono, son distribuidos, actúan en paralelo y/o son no deterministas.

Las redes de Petri realizan una abstracción a nivel funcional. Su modo de operación es no determinista y permiten reflejar, entre otros aspectos, la ejecución concurrente de distintos procesos, representar la disponibilidad de recursos, imponer restricciones de acceso a datos compartidos, y representar la evolución dinámica de los sistemas. Extensiones posteriores al modelo inicial permiten manejar el tiempo, asignar prioridades de actuación, estructuración jerárquica, etc.

Las redes de Petri se han utilizado para expresar la semántica de los workflows que son un modelo de la logística de los procesos del negocio. Un proceso de workflow es realmente un diagrama que especifica el flujo de tareas que se deben ejecutar en un orden determinado y bajo condiciones determinadas.

Proyectos donde se han aplicado las Redes de Petri.

Se realizó un estudio de Verificación de Programas de PLCs Modelados sobre Redes de Petri. Métodos Gráficos vs Algebraicos en la Empresa de Telecomunicaciones de Cuba S.A. (ETECSA) en Las Tunas mostrándose dos poderosos métodos de análisis y verificación de programas de PLCs modelados sobre redes de Petri (PN). Ambos métodos han sido aplicados en automatizaciones docentes e industriales, como es el caso de las instalaciones de laboratorio utilizadas en la carrera de Ingeniería en Automática y en Cursos de Postgrado de la Facultad de Ingeniería Eléctrica, Universidad de Oriente, así como en la automatización realizada en las terminales de embarque de azúcar a granel de los puertos cubanos de Carúpano, Guayabal,

Mariel y Cienfuegos. Las PN constituyen el método formal de mayores posibilidades en este campo.

Ventajas de las Redes de Petri.

La principal ventaja de las Redes de Petri es su capacidad para el análisis de gran cantidad de propiedades y aspectos ligados a los sistemas concurrentes. Además de que permiten modelar sistemas donde un recurso es compartido posibilitando mayor facilidad de comunicación. También cuenta con un tratamiento individual de procesos independientes y opera con procesos concurrentes.

Herramientas que utilizan

CODESIGN, ALPHA/Sim, Artifex, Design/CPN HiQPN, INCOME, Netmate, PEP, PNTalk, Thorn/DE.

2.4 Métodos de Especificación Formal algebraicos.

Los métodos de Especificación Formal algebraicos son en los que las operaciones se definen mediante relaciones de equivalencia (con la igualdad en general). Estos métodos son fundamentalmente útiles para especificar interfaces entre los componentes del sistema.

2.4.1 Larch

Larch es un proyecto de investigación cuyo objetivo era explorar métodos, lenguajes y herramientas para facilitar el uso de las especificaciones formales. El proyecto se inició a principios de los 80 en el Laboratorio de Ciencias de la Computación del MIT y en el Centro de Investigación de Sistemas de Digital en Palo Alto, California. Su principal característica es que

contiene dos estilos de especificación. Este método es especializado en la especificación del comportamiento de sistemas secuenciales.

Cada especificación se escribe en dos lenguajes:

- Lenguaje de interfaz Larch: Diseñado para un lenguaje de programación específico. Se utiliza principalmente para describir las interfaces entre los componentes del programa a través de tipos abstractos de datos.
- Lenguaje compartido Larch (Larch Shared Language): Es independiente del lenguaje de programación. La unidad básica de una especificación Larch se denomina trait e incluye dos tipos símbolos: operadores y conjuntos. Es muy similar a otros lenguajes de especificación algebraicos.

Herramientas

El demostrador de teoremas Larch Prover y LCLint son dos de las herramientas desarrolladas en el proyecto Larch.

Conclusiones.

Se puede decir que los Métodos Formales, en sí, tienen una gran importancia aplicados a la especificación de requisitos ya que todos de una manera u otra pueden ser usados de forma efectiva para solucionar el problema determinado según sus características, que pueden ser frente a algunos ventajas pero frente a otros son limitaciones ya que un solo Método Formal no satisface todas los requisitos que un sistema requiera.

CAPITULO 3: PROPUESTA.

Introducción

En este capítulo se hace un análisis crítico de los métodos formales estudiados en el capítulo 2 para conformar la propuesta que se dará al culminar el mismo. Se explica por qué se excluyeron algunos métodos. Así como los aspectos que sirvieron de apoyo para escoger los métodos formales que conformaran la propuesta.

3.1 Métodos Formales que se excluyen.

Los criterios que se tuvieron en cuenta para excluir algunos métodos formales de la propuesta obtenida fueron los siguientes:

- Aquellos que sean muy difíciles de aplicar por sus características.
- Difíciles de aplicar por las característica de los proyectos de la Facultad3.
- Los que las herramientas que utilizan los proyectos de la Facultad 3 no son capaces de soportarlos.
- Aquellos métodos cuyas ventajas estén incluidas en otros métodos formales más las de ellos en específico.

Las Máquinas de Estados Finitos.

Las Máquinas de Estados Finitos, a pesar de ser una técnica antigua de representación de conocimiento y modelado de sistemas, y de ser usadas desde hace tiempo, y de las cual se puede aprender mucho, son muy difícil de aplicar ya que no pueden utilizarse para proyectos que implementen un sistema grande, se le haría difícil la administración y la manutención de un buen diseño.

También no es apropiada para todos los dominios de un problema, solo debe ser usada cuando el comportamiento de un sistema puede ser descompuesto en estados separados con condiciones bien definidas para las transiciones. Lo que significa que todos los estados, transiciones y condiciones deben ser conocidos y estar bien definidos. Los métodos basados en máquina de estado finito son SDL y ESTELLE.

SDL (Specification and Description Languange)

El método SLD que se especializa en la descripción de sistemas concurrentes. A pesar de que este método brinda mucha utilidad en la especificación de requisitos, describe el sistema como un conjunto de máquinas de estados finitos lo que lleva a que tenga las mismas las mismas limitaciones del método máquinas de estados finitos, con la diferencia de que este se representa en bloques, lo que como toda creación esta perspectiva tiene su ventajas ya que pueden comunicarse entre ellos. Solo son buenos para aplicarlos en aquellos sistemas que sean basados en estímulo-respuesta.

ESTELLE (Extended State Transition Language).

EL método ESTELLE es muy difícil de aplicar porque en un sistema se especifica mediante un conjunto de máquinas de estados finitos que se comunican entre sí mediante colas y variables globales. En estos casos los analistas corren el riesgo de no determinar qué políticas o estrategias de la empresa son la guía para la toma de decisiones específicas.

Larch.

Larch es un proyecto que creó dos tipos de especificación: Lenguaje de interfaz Larch y Lenguaje compartido Larch, el primero solo para un lenguaje de programación específico y las características del segundo son muy similar a otros lenguajes de especificación algebraicos. Este método presenta problemas para la verificación formal de la completitud de las especificaciones. Precisamente lo que se aspira en este trabajo es que se mejore las especificaciones de los requisitos en el desarrollo del software. Por lo que este método queda excluido del sistema.

La lógica temporal.

La Lógica Temporal no pertenece a la clasificación de técnicas no constructivas que son las que son orientadas a propiedades que se aplican en la fase inicial de especificación de requisitos, sino que corresponde a las constructivas por lo que su aplicación fundamental es para la parte de diseño por lo que no seria una buena opción utilizarlo en la especificación de requisitos, más bien sería un riesgo pues se pudieran introducir errores que con otros métodos nos evitaríamos.

CSP (Calculus of Sequential Processes)

El método CSP se excluye porque la restricción de su aplicación a sistemas secuenciales se eliminó entre 1978 y 1985. También porque sus características están expuestas en Z que es uno de los métodos que se escogió.

Redes Petri.

Las Redes de Petri tienen muchas ventajas al representar una gran variedad de sistemas, al igual que otros métodos como lo diagramas de flujos realizan una abstracción a nivel funcional, pero es una clase particular de grafo dirigido por lo que los autores determinan que especificar los requisitos de un sistema como tal sería muy engorroso. Además las características que poseen respecto a que son buenos en sistemas distribuidos las tiene el método LOTOS, que ve un sistema distribuido como un proceso. Por lo que este Método formal se excluye de la propuesta.

VDM (Vienna Development Method)

A pesar de ser una colección de técnicas para la especificación formal y el desarrollo de sistemas software, el método VDM no se utiliza mucho en la etapa de análisis pero si la etapa de diseño. Los métodos VDM y Z han sido utilizados en un mismo proyecto. Además como los dos están basados en teorías de conjuntos y lógica de primer orden, el método Z cumple con las mismas funciones que VDM en el sentido de que ambos se pueden utilizar para aquellos sistemas secuenciales y ha resultado más recomendado ya que es el más utilizado a nivel mundial.

TRIO+.

El método TRIO+ no se incluye en la propuesta porque precisamente se hizo con el objetivo especificar sistemas empotrados y de tiempo real lo que no cumple con los objetivos de este trabajo, ya que en esta propuesta se buscan métodos que estén diseñados para especificar proyectos de gestión que son en los que trabaja la facultad 3.

3.3 Aspectos a considerar para la utilización de Métodos.

Dada la situación de que ningún método brinda la mejor solución para todos los problemas, es necesaria la utilización de diversos tipos de métodos. Para que los mismos tengan éxito entre sus posibles usuarios deben tenerse en cuenta las siguientes características:

- Rápido retorno de la inversión: Las personas que utilicen estos métodos deben ser beneficiados desde el primer momento que lo utilicen.

- Los beneficios deben incrementarse con el esfuerzo: A medida que los desarrolladores aumenten sus esfuerzos en escribir especificaciones o utilizar las herramientas, los beneficios que aportan estos métodos formales deben hacerse mayores.

- Múltiple uso: La utilización de los métodos formales es mayor si este tiene varios usos.

- Uso integrado: Los métodos deben integrarse perfectamente con las técnicas y lenguajes de programación que utilicen los proyectos de la facultad, de forma que en vez de empeorar las cosas las mejore, es decir, que los desarrolladores no tengan que adaptarse totalmente a una nueva tecnología para sacar partido de estos métodos.

- Facilidad de uso: Esto aumenta el atractivo de un método ya que como es fácil de usar, se obtendrá mejores resultados y se ganará en tiempo.

- Eficiencia: Un método formal debe ser eficiente para que su utilización no sea una pérdida de tiempo.

- Fácil de aprender: El tiempo necesario para aprender el manejo de los métodos o herramientas debe ser el menor posible, lo que posibilitara una mayor aceptación de los mismos por parte de los desarrolladores y el retorno de la inversión será más rápido.

- Orientación a la detección de errores: Los métodos y las herramientas deben estar enfocados, más a la detección de errores, que a la certificación de la corrección de un sistema.

- Análisis focalizado: Los métodos y herramientas deben ser buenos analizando al menos uno de los aspectos del sistema.

3.2 Propuesta de un Sistema de Métodos Formales para la Especificación de Requisitos para los proyectos de la facultad 3.

Los métodos formales no son aplicables a cualquier tipo de desarrollo de software porque son muy difíciles de aplicar y por eso se sugiere en las mayorías de la bibliografías que se apliquen en aquellos software que sean empotrados, a sistemas en los cuales se manejan vidas humanas, (sistema de control de vuelos, sistemas de tiempo real de plantas nucleares, sistemas médicos), porque en estos sistemas se necesita la mayor fiabilidad posible. Además utilizar métodos formales requiere de un coste muy elevado ya que se debe contar con un ingeniero experto en la materia, que se capacite a los analistas en la notación formal y se debe contar con una persona que tenga buenos conocimientos matemáticos que le pueda transmitir al cliente como es que quedan sus necesidades reflejadas en la especificación.

Lo antes expuesto motivó a los autores a la selección de los Métodos Formales que conformaría la propuesta donde los mismos se puedan adecuar a las características de la facultad y se pudieran aplicar a software de gestión. Por lo que los autores proponen que la Propuesta de un Sistema de Métodos Formales para la especificación de requisitos en la facultad 3 estará conformada por OCL como base, Z y Lotos**.** Esta propuesta mejoraría grandemente el proceso de desarrollo del software y la calidad del producto final ya que las especificaciones no serian incompletas, inconsistentes o engañosas por lo que el la fecha de entrega y el alcance del software no sufrirían consecuencias adversas.

3.4 ¿Por qué utilizar OCL como base?

Con el empleo del Método Formal OCL se consigue, que la especificación pueda ser analizada formalmente, es decir, que pueda ser probada su consistencia así como ciertas propiedades del sistema especificado. Este método viene siendo como el punto intermedio entre los lenguajes naturales y los formales a pesar de catalogado como tal, por lo que su uso es más fácil que los métodos formales tradicionales. Se desarrolló para llenar el vacío que deja el uso del lenguaje natural cuando se utiliza para definir esas condiciones o semánticas adicionales que son requeridas en todo modelo, uso que las mayorías de las veces causa ambigüedades en el modelo, y para aquellos que tienen una fuerte base matemática, que dificultan su uso en las empresas y modeladores de software.

Como OCL es componente de UML, éste puede utilizar las funciones que brinda UML, tal es el caso que permitió anteriormente excluir métodos que cumplían con estas funcionalidades pero que presentaba otras limitaciones. Además hay diagramas de UML, como es el caso de los diagrama de clases, que normalmente no están lo suficientemente refinado por lo que no se refleja todos los aspectos relevantes de una especificación, por lo que OCL a través de las restricciones que hace en todas sus variantes le brinda solución a esto.

Hay herramientas con las que trabajan los proyectos de la facultad 3 que lo soportan, por lo que sería un paso de avance al no tener que incursionar en algo diferente y no sería empezar desde cero. Este no llega a ser considerado para muchos un método formal de los más fuertes por lo que se ha dicho anteriormente respecto a su base matemática porque sus diseñadores decidieron que este solo debería usar caracteres ASCII lo que no quiere decir que no disponga de todo el poder de la lógica y las matemáticas discretas, a pesar de ser una debilidad de este método a la vez suele ser una ventaja ya que permite que el leguaje sea más asequibles a las personas menos inclinadas a las matemáticas y que las computadoras lo procesan con mayor facilidad.

La semántica de OCL se aplica sobre la base de la construcción de herramientas CASE que soportan comprobaciones de integridad sobre todos los modelos UML, no sólo sobre los componentes expresados en OCL.

Ejemplo: Si se quiere expresar el límite del tamaño de un equipo de menos de diez miembros. Con el OCL, usted puede declarar esta regla del negocio como un invariante del contexto.

Context Team **inv**:

self.numberOfMembers <= 10

Otro ejemplo seria:
Cuando se quiere representar que toda compañía tiene empleados.

Context Compañía inv:
self.empleado→notEmpty ()

El propósito de OCL es aportar formalidad de manera clara y legible. Es un método con el que es posible evitar grandes párrafos de lenguaje natural que podrían ser confusos. Z y Lotos

vienen siendo métodos con base matemática más sólidas que se utilizaran en la especificación de requisitos críticos que necesiten de las características de estos.

¿Por que se escogen Z y LOTOS?

Notación Z.

- Fuerte base matemática lo que le permite modelar el sistema mediante entidades matemáticas conocidas, tales como conjuntos, relaciones, funciones y secuencias.
- Permite la construcción de modelo de forma incremental.
- Modela sistemas secuenciales.
- Modelación de estructuras de datos.
- Permite la descripción abstracta del efecto década operación en un sistema, los invariantes que deben cumplirse y el razonamiento sobre el comportamiento del sistema

LOTOS (Language of Temporal Ordering Specification).

- Posibilita la especificación formal de los sistemas distribuidos.
- Facilita el modelo de sistemas concurrentes.
- Permite que la especificación 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.

- Su semántica formal permite realizar descripciones completas, claras, consistentes y no ambiguas del sistema a especificar.

- Tiene una fuerte base matemática que la sustenta lo que posibilita el desarrollo y aplicación de métodos de validación y verificación de propiedades y equivalencias.

- No solamente garantiza que la especificación cumpla 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.

- Su empleo ayudaría a descubrir, más rápidamente, la existencia de posibles errores o falta de funcionalidad en la especificación del sistema.

Métodos Formales Incluidos.	Proceso de especificación.
OCL (Object Constraint Language)	Después de realizado el proceso de obtención de requisitos se procede en el modelamiento del negocio a obtener las reglas del mismo las cuales son los requisitos expresados de manera mas general, aquí es donde entra en escenario OCL: 1. Se definen el invariante de datos. 2. Se determinan el estado. 3. Se determinan las operaciones que se asocian a dos condiciones pre-condición u pos-condición.

Notación Z	Teniendo una especificación informal se procede a: 1. Definición de tipos donde se declaran los tipos básicos en dependencia al sistema que se esta especificando. 2. Descripción de estados abstractos esto se hace para tener un medio donde registrar los tipos básicos, que a medida que se inserte uno nuevo entonces el sistema cambiará de estado. 3. Se determina el estado inicial del sistema. 4. Se realizan operaciones parciales donde se hacen una serie de pruebas que permiten asegura que la especificación corresponde con lo que se requiere. 5. Se realizan operaciones totales ya al final que son las que ayudan a mejorar los errores. Es importante mencionar este proceso forma parte del proceso de construcción de un sistema en términos de un programa ejecutable, permitiendo conseguir un documento que pueda leerse y comprenderse con mayor facilidad.

Tabla 3. Métodos formales Incluidos

3.5 Herramientas.

A pesar de que en el subepígrafe **3.1** se expresa que unos de los aspectos que se tuvieron en cuenta para excluir métodos formales fue que los proyectos de la facultad 3 no soportaran las herramientas, es decir, que no las utilizaran, no fue del todo así ya que como no tenía conocimientos de los métodos formales tampoco se conocían las herramientas que estos utilizaban pero al realizar este estudio, los autores recomiendan que se utilice Enterprise Architect y para minimizar sus debilidades, Zans y NOTOS .

3.5.1 Enterprise Architect.

Enterprise Architect es una herramienta CASE (Computer Aided Software Engineering) para el diseño y construcción de sistemas de software, soporta la especificación de UML 2.0, que describe un lenguaje visual por el cual se pueden definir mapas o modelos de un proyecto.

Según el manual de esta herramienta:
Enterprise Architect es una herramienta progresiva que cubre todos los aspectos del ciclo de desarrollo, proporcionando una trazabilidad completa desde la fase inicial del diseño a través del despliegue y mantenimiento. También provee soporte para pruebas, mantenimiento y control de cambio. Esta herramienta tiene la ventaja que se puede ajustar a los diferentes roles que desempeñan los integrantes de un proyecto para realización de un sistema.

Enterprise Architect en el Modelado de Requisitos que es muy importante para la implementación de un proyecto, permite a los usuarios definir elementos del tipo de requisitos, requisitos vinculados a los elementos del modelo de implementación, requisitos vinculados y su jerarquía, reportes de requisitos, y mover requisitos dentro y fuera de la responsabilidad de los elementos del modelo.

Tiene una parte un diálogo de Configuración de la Validación del Modelo en cual se puede especificar las características que uno desee establecer. Se puede acceder a este mediante

Proyecto | Validación del Modelo | Configurar. Para mayor información leerse el manual de esta herramienta.

3.5.3 Zans.

Zans es una herramienta de animación para las especificaciones en Z. Es un prototipo en evolución. La versión con la que se trabaja en la práctica es la 0.31. Los objetivos de Zans son:

- Facilitar la validación de las especificaciones en Z
- Experimentar en refinamiento de diseño y síntesis de código basada en especificaciones Z
- Ayudar en el aprendizaje del lenguaje Z

Esta herramienta trabaja utiliza para cumplir sus funcionalidades una serie de comandos ya definidos como son:

- Para cargar una especificación Z se utiliza: load <nombre_archivo>
- Para animar la especificación Z se utiliza: animate
- Para ejecutar una operación de la especificación: execute <nombre_operacion>
- Para mostrar un listado de los esquemas que forman parte de la especificación: list
- Para visualizar un esquema: show <nombre_esquema>

Por lo antes explicado se recomienda que se utilice Zans para especificar requisitos en sistemas que tengan construcciones matematicas muy fuertes.

3.5.4 NOTOS.

El generador de prototipos NOTOS analiza una especificación de un sistema escrita en el subconjunto del LOTOS, 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.

Esta se compone de dos grandes fases o bloques: la fase de análisis y la fase de generación del prototipo. En la fase de análisis se estudia que la especificación de entrada sea sintácticamente correcta. Para ello, se implementan las distintas fases de análisis de un compilador. NOTOS, fuera capaz de realizar un seguimiento a distintos niveles de las expresiones del comportamiento.

Se recomienda el uso de la herramienta antes explicada para realizar especificaciones en sistemas que sean distribuidos y concurrentes contribuyendo a la mejora de la calidad en esta fase de desarrollo del software.

3.6 Métricas de la calidad de la especificación.

Tomando como partida unos de los mandamientos de los métodos formales en los que se hizo alusión en el capitulo1, precisamente en el VII, en que no se deben comprometer los estándares de calidad, ya que estos no son del todo perfecto, se aconseja que aunque se utilicen en estos métodos se haga uso de métricas.

Existen propuestas (PRESSMAN 2006) un grupo de características que pueden ser empleadas en apreciar la calidad del modelo de análisis y la especificación de requisitos: especificidad (ausencia de ambigüedad), compleción, corrección, comprensión, capacidad de verificación, consistencia interna y externa, capacidad de logro, concisión, trazabilidad, capacidad de modificación, exactitud y capacidad de reutilización. Los autores apuntan que las especificaciones de alta calidad deben estar almacenadas electrónicamente, ser ejecutables o al menos interpretables, anotadas por importancia y estabilidad relativas, con su versión correspondiente, organizadas, con referencias cruzadas y especificadas al nivel correcto de detalle. Aunque muchas de las características anteriores parecen ser de naturaleza cualitativa,

Davis sugiere que todas puedan representarse usando una o más métricas. Por ejemplo asumimos que hay nr requisitos en una especificación tal como:

nr = **n**f + **n**nf

Donde **n**f es el número de requisitos funcionales y **n**nf es el conjunto de requisitos no funcionales (por ejemplo el rendimiento).
Para determinar la especificidad (ausencia de ambigüedades) de los requisitos. Davis sugiere una métrica basada en la consistencia de la interpretación de los revisores para cada requisito:

Q1 = **n**ui / **n**r

Donde **n**ui es el número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas. Cuanto más cerca de 1 este el valor de Q, menor será la ambigüedad de la especificación. La compleción de los requisitos funcionales puede determinarse calculando la relación

Q2 =**u**n/ (**n**i*ns)

Donde **u**n es el numero de requisitos únicos de función, **n**i es el numero de entradas (estímulos) definidos o implicados por la especificación y **n**s es el numero de estados especificados. La relación Q2 mide el porcentaje de funciones necesarias que se han especificado para un sistema. Sin embargo, no trata los requisitos no funcionales. Para incorporarlos a una métrica global completa, debemos considerar el grado de validación de los requisitos.

Q3= **n**c / (**n**c + **n**nv)

Donde **n**c es el numero de requisitos que se han validado como correctos y **n**nv el numero de requisitos que no se han validado todavía.

3.7 Opiniones de especialistas.

La propuesta de un Sistema de Métodos Formales para la Especificación de Requisitos para los proyectos de la facultad 3 fue valorada por varios especialistas teniendo en cuenta la importancia que tiene aplicar los métodos formales en la Etapa de Desarrollo de la Ingeniería de Requisitos.

Esta propuesta es importante para la introducción de los métodos formales en proyectos productivos de la facultad 3. Para ello se debe lograr una mayor vinculación con la asignatura de Matemática Discreta, y adoptar el enfoque llamado “Lightweight Formal Methods”. Con los métodos formales se eliminan las ambigüedades propias de los lenguajes humanos.

Msc. Maikel Yelandi Leyva.
Profesor Ingeniería de software.
Graduado en informática. CUJAE/UHO.
Profesor instructor
2 años de experiencia.

Si el alcance de la solución propuesta, a la no existencia de formalidad en el planteamiento de los requisitos en la UCI, está planteado que es solo para los tipos de software cubiertos por las métodos OCL, LOTOS y Z: la considero una buena propuesta, necesaria para la universidad y útil teniendo en cuanta que no se ha definido una manera estándar de describir los requisitos.

Además la propuesta no se involucra con el levantamiento de la información del cliente solo con la verificación de la información gestionada, tarea que entiendo que es realizada de una manera correcta, además brinda la posibilidad de una vez que el cliente entienda la sintaxis del método validar las reglas del negocio evitando imprecisiones y ambigüedades.

Solo resta sugerir que propongan este conocimiento como parte de las asignaturas de pre-grado para que el estudiantado tenga algunas bases que le permitan en el futuro asimilar este

conocimiento, pues por las funciones que realizo en IP considero que el describir bien, determinar y validar las necesidades de los clientes es una de las tareas más importantes de la ingeniería para lograr un producto de calidad y se además que no es la característica fuerte de los actuales proyectos.

Ing. Michael González Jorrin
Graduado de: Ing informática
Ocupación: Especialista Dir Calidad y Normas
Años de trabajo en la UCI: 5
Experiencia en pruebas a software:

1. Registros y Notarías
2. Identidad
3. Digitalización
4. Multimedia en General (Che, Constitución Venezolana, ECOSOL, etc.)
5. Intranet de PDVSA
6. Pruebas de aceptación en el extranjero

Eventos científicos en que ha participado:

1. Curso avalado de CMMI
2. Evento Sepgla 2004
3. Evento Metánica de calidad
4. Informática 2007
5. Uciencia

Esta propuesta de un Sistema de Métodos Formales para la Especificación de Requisitos en los Proyectos de facultad 3, es una alternativa más posible de utilizar en la Etapa de Desarrollo de la Ingeniería de Requisitos. Realmente los métodos formales son difíciles de entender y aplicar, no creo que tengan aplicación generalizada, aunque la selección de OCL, componente de UML, puede atenuar esta situación y facilitar la interacción con el cliente. Por eso debe recomendarse

su aplicación en proyectos internos donde tanto el cliente como el desarrollador son profesionales o estudiantes de la informática.
MSc. Eugenia G. Muñiz Lodos
Prof. Titular Adjunta
Investigador Auxiliar ICID
Más de 30 años de experiencia desarrollando software.

Esta propuesta de Sistema de Métodos Formales para la Especificación de Requisitos en los Proyectos de facultad 3, es una buena opción ya que serviría de guía a los integrantes de estos proyectos cuando se vayan a involucrar en el estudio de este tema avanzado de la Ingeniería de Software ya que no empezarían desde cero. Además es bueno recalcar que con la utilización de estos en la Etapa de Desarrollo de la Ingeniería de Requisitos se evitarían cambios en el sistema ya que los requisitos quedarían sin ambigüedades y evaluados de una forma matemática.

Ing. Rolando Pérez Pinto
Profesor de Gráfico por Computadora.
Profesor instructor.
Graduado en informática CUJAE.
1 año de experiencia.

Conclusiones.

En este capítulo se determinó que:
Los métodos formales tienen diferentes funcionalidades y no son aplicables a cualquier proyecto Y que independientemente de que se les de su mejor uso se deben aplicar métricas de especificación para lograr una calidad completa.

Con un solo método formal aplicado no es suficiente para abarque y satisfaga todas las funcionalidades de un sistema, ni cubra las necesidades de los clientes por que se pueden

combinar varios métodos formales para lograr una especificación completa. Por lo que la utilizar el sistema de los métodos formales integrado por OCL, Z y LOTOS en cuanto a la especificación de requisitos es, según el estudio realizado, adaptable a las condiciones específicas de los proyectos de la facultad 3.

CONCLUSIONES

Para el desarrollo de este Trabajo de diploma se llevó a cabo una exhaustiva búsqueda bibliográfica que llevó a un profundo estudio de los Métodos Formales para la Especificación de Requisitos, en el mismo se expuso la necesidad de proponer un Sistema de Métodos Formales para la Especificación de Requisitos acorde a las características de los proyectos de software de la facultad 3, que por lo general son de gestión, que brindará la posibilidad de mejorar la fase de especificación de requisitos. Primeramente se hizo un análisis sobre lo que se hacia en esta fase y a partir de la problemática encontrada, se realizó un estudio de los métodos formales desarrollados para especificar requisitos y se escogieron, a partir de un análisis crítico, aquellos que mejor respondían a los intereses de los proyectos de la facultad 3.
Partiendo de esta investigación se elaboró la propuesta de utilizar OCL, LOTOS y Z para la especificación de requisitos que permitirá producir software con mayor calidad.

Por todo lo anteriormente dicho se considera cumplido el objetivo planteado en esta investigación.

RECOMENDACIONES.

1. Incitar al uso de los métodos formales en los proyectos de la facultad 3 e ir creando la cultura del uso de los mismos debido a la importancia mundial que tiene en el campo de las ciencias informáticas. Esto permitirá que la facultad 3 escale en los temas avanzados de la ingeniería del software.

2. Que se capacite a los profesores de la especialidad en este tema de la ingeniería del software o al menos que la facultad cuente con un experto en métodos formales.

3. Que se impartan cursos a los analistas de los proyectos que tienen que hacer la especificación de requisitos.

4. Capacitar a una persona que sirva como interlocutor es decir que este del lado del cliente, que tenga buenos conocimientos matemáticos y le pueda transmitir toda la información de manera entendible.

5. Continuar profundizando en el estudio y enriquecimiento de la utilización de varios métodos formales. Validar esta propuesta empíricamente lo que permitirá depositar una mayor confianza en los resultados que se obtengan de la aplicación de los métodos formales para la especificación de requisitos en los proyectos de la facultad 3.

REFERENCIAS BIBLIOGRAFICAS.

BIGUS, J. P. B. J. Constructing Intelligent Agents Using Java - Second Edition. 2001. p.

CHANDY, K. M. Y. K., C. The Derivation of Compositional Programs. En Proc. of the 1992 International Joint Conference and Symposium on Logic Programming. MIT Press., 1992.

DUKE, J. G. Especificacion, Verificación y Mantenimiento de Requisitos Funcionales con Técnicas de Descripción Formal. Dpto. de Tecnologías de las Comunicaciones ETSI de Telecomunicación. Universidad de Vigo, 2000. p.

GOTZHEIN, R. Temporal logic and its applications. Computer Networks and ISDN Systems. 1992. p.

HAYES I.J., J. C. B. Specifications are Not (Necessarily) Executable.Technical Report Nro.148. Departament of Computer Science, University of Queensland, 1990. p.

JONES, C. B. Systematic Software Construction Using VDM. Prentice Hall International Series, 1990.

MORZENT, M. F. Y. A. Validating Real Time Systems by History-checking TRIO Specifications. ACM Transactions on Software Engineering and Methodology. 1994. p.

PNUELI, A. Linear and branching structures in the semantics and logics of reactive systems. In 12th ICALP. 1985. p.
---. The Temporal Logic of Programs. En Proc. of the 18th Annual Symposium on the Foundations of Computer Science, 1977. pp. 46–57. IEEE.

HOARE. Communicating Sequential Processes. 1967. p.

PRESSMAN, R. Ingeniería de Software. Un Enfoque Práctico. Sexta Edición, 2006.
---.Ingeniería de software: Un enfoque práctico Quinta Edición. La Habana, Editorial Félix Varela. , 2005. p.
---.Ingeniería de software: Un enfoque práctico Quinta Edición. La Habana, Editorial Félix Varela. , 2002. p.

SOLLA, A. G. Diseño y Verificación de Sistemas Distribuidos mediante la aplicación Combinada de Métodos Formales, 1999.

SPIVEY, J. M. The Z Notation. Prentice Hall series in computer science. 1992. p.

BIBLIOGRAFIA.

http://www.di.uniovi.es/~cernuda/pubs/tesis.pdf (Mayo 2007)

http://vidaartificial.com/index.php?title=Maquinas_de_Estados_Finitos_%28ai-depot.com%29 (Mayo 2007)

http://www.unlu.edu.ar/~ogarcia/ia/doc/Aprendizaje.ppt (Abril 2007)

http://www.unlu.edu.ar/~ogarcia/ia/doc/Aprendizaje.ppt (Abril 2007)

http://lotos.csi.uottawa.ca/ftp/pub/Lotos (Mayo 2007)

http://www.dit.upm.es/~lotos/ (Mayo 2007)

The Formal specification language LOTOS U. Of Stirling, Kenneth **J. Turner**. (Mayo 2007)

Ingeniería de Software. Un Enfoque Práctico. Sexta Edición, **Roger Pressman**. (Mayo 2007)

Ingeniería de software: Un enfoque práctico Quinta Edición, **Roger Pressman** (Abril 2007)

GLOSARIO DE TERMINOS.

Herramientas CASE: Las Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) tienen varias aplicaciones informáticas dirigidas a aumentar la productividad en el desarrollo de software, ayudan en todo el ciclo de vida del desarrollo del software tanto en tareas como el proceso de realizar un diseño del proyecto.

Proceso: Un proceso de ingeniería de software es una definición del conjunto completo de actividades necesarias para transformar los requisitos de usuario en un producto final.

Proceso de Desarrollo de Software: Es el conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto.

Procesos concurrentes: Son aquellos donde se ejecutaran una serie de procesos al mismo tiempo, y frecuentemente con un grado elevado de comunicación entre ellos.

Procesos secuenciales: Son aquellos programas simples no concurrentes que se ejecutan y se comunican de manera autónoma.

Componentes: Es un conjunto o bloque de software que proporciona por si mismo o en conjunción con otros componentes una función o servicio único y puede ser re-utilizado para construir diversos sistemas.

Sistemas reactivos: como sistemas de computación que están continuamente interactuando con el entorno, de tal manera que tienen que actuar de inmediato ante los estímulos por éste producidos.

Sistema distribuido: Es aquel sistema que esta formado por varios sistemas autónomos que trabajan de forma coordinada para la realización de una determinada tarea. Estos sistemas suelen estar compuestos de un conjunto de procesos concurrentes que interactúan entre sí. Se

caracterizan más por reaccionar a estímulos del entorno que por terminar produciendo algún tipo de resultado final.