# Universidad de las Ciencias Informáticas
## Facultad # 2


# Título: Aplicación de los métodos formales en la Ingeniería de Software.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

**Autores:** Patricia López Mass.

William Sánchez Abreu.

**Tutor:** Ing. Maypher Román Durán

Julio 2007

## DECLARACIÓN DE AUTORÍA

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

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

Patricia López Mass

_____

Firma del Autor

William Sánchez Abreu

_____

Firma del Autor

Ing. Maypher Román Durán

_____

Firma del Tutor

**Datos de Contacto**

Ing. Maypher Román Durán

Correo electrónico: maypher@uci.cu

**Pensamiento**

“El mundo camina hacia la era electrónica…
Todo indica que esta ciencia se constituirá en
algo así como una medida del desarrollo;
quien la domine será un país de vanguardia.”

*Ernesto Ché Guevara.*

**Agradecimientos**

A mi mamá por darme su amor incondicional.

A mi papá por el apoyo que me ofrece.

A mi hermana por todo su cariño.

A mis abuelos por esperar siempre lo mejor de mí.

A mi tutor por su entrañable ayuda.

A todos los profesores que dedicaron su tiempo en nuestra formación.

A mis amigos y mis compañeros de aula.

A todos los que se preocuparon y estuvieron siempre al tanto de mi formación.

A todos los que me ofrecieron su ayuda desinteresadamente.

A todos los que de una forma u otra contribuyeron al desarrollo de este trabajo.

*Patricia López Mass*

A mis padres especialmente por brindarme ese apoyo incondicional en todo momento.

A mi tutor por su ayuda a la realización de este trabajo.

A todos los profesores que dedicaron su tiempo en nuestra formación como estudiante y persona.

A mis compañeros de aula y de universidad, a todos aquellos compañeros que aunque no están en la UCI me brindaron su apoyo.

A esas personas que me han soportado durante todo el transcurso de la universidad, en especial a las muy cercanas a mí.

A todos los que me ofrecieron su ayuda desinteresadamente.

A todos los que de una forma u otra contribuyeron al desarrollo de este trabajo.

*William Sánchez Abreu.*

**Dedicatoria**

*A mis padres, mi hermana, mi sobrina y a toda mi familia.*

*Patricia*

*A mis padres, amigos y amigas más apegadas a mí.*

*William*

**Resumen**

En este trabajo de diploma se hace un estudio de los métodos formales estudiados en otros países del mundo, así como algunos lenguajes de especificación formal que existen y herramientas para el trabajo con dichos métodos con el objetivo de crear una estrategia de trabajo para aplicar los métodos formales en la producción de software en la universidad. Esta técnica ofrece muchas ventajas en la modelación y producción de software, dada su capacidad de proporcionar un software más seguro y con menos errores en su desarrollo, logrando una mayor calidad en los mismos. Para esto se investigó sobre los temas relacionados con estos métodos y que se imparten en otras asignaturas de la carrera y se vinculan con los temas tratados en la ingeniería de software desglosándolo por cada flujo de trabajo. Además, desde el punto de vista de la producción se analizan algunos de los proyectos que se están realizando en la UCI y se establece si es factible o no, que se utilicen en ellos los métodos formales, para ello también se establece cuáles son los integrantes de un equipo de desarrollo que deben profundizar los conocimientos formales según el rol y la etapa en que se desempeñan.

**Índice**

Índice

## Introducción

Una parte importante de la capacidad del desarrollador de software esta dada por la habilidad que tenga este para evaluar las características de las herramientas disponibles para la producción de software. Para esto es necesario que cuente con bases profesionales sólidas para poder analizar efectivamente la posibilidad de aplicación económica de la tecnología CASE, generadores de código, software de base y otros ambientes relacionados con la producción de software. Todas estas bases profesionales estarán basadas, principalmente, en los métodos formales sobre los que se sustenta la Ingeniería de Software.

Se trata de técnicas matemáticas basadas en la especificación y que analizan, definen y hacen pruebas de software de una forma muy rigurosa y se figura que su uso para el diseño del software y del hardware es motivado por la expectativa que, como en otras disciplinas de la ingeniería, la ejecución de análisis matemáticos apropiados puede contribuir a la confiabilidad y a la robustez de un diseño.

Históricamente, los métodos formales han centrado sus objetivos en obtener un software con una buena calidad, y han descuidado, en gran medida, el entorno en el que debían aplicarse. La falta de una forma eficiente para su estudio y el déficit de herramientas para su aplicación los convirtieron en recursos bastante ideales y difíciles de manejar. Su utilización se ve limitada por su alto coste y solo se aplican a sistemas de alta integridad donde se necesita un alto nivel de seguridad.

Pero sin duda, la sistematización progresiva de la producción de software necesita de notaciones expresivas y, evidentemente formales. No existe la automatización sin formalización.

Desde su inauguración la Universidad de las Ciencias Informáticas (UCI) ha hecho énfasis en el desarrollo de software, estudiando formas y técnicas para la modelación de los mismos. Con este propósito la universidad cuenta con un grupo de profesores que se encargan de impartir la asignatura de Ingeniería del Software con un plan de clases que cubre contenidos sobre esta materia.

Introducción

El departamento de esta especialidad se dio a la tarea de investigar nuevas formas de modelado con el objetivo de conocer y comprobar la eficiencia y eficacia de estas técnicas al aplicarlas a los software que se desarrollan. Esta nueva variante son los métodos formales en la Ingeniería de Software, una técnica poco utilizada y muy poco conocida en Cuba y que pueden constituir una gran fuente de ayuda para desarrollar todos los proyectos de software con una mayor calidad y robustez.

Se tiene la situación problémica de que en Cuba el estudio de los métodos formales está muy poco difundido, sólo se estudian algunos de sus temas, incluidos en los programas de otras asignaturas de la carrera de Ingeniería Informática (Matemática Discreta, Introducción a la Programación, etc.) y muchos de ellos de una forma muy diferente a cómo se estudian y se aplican en las universidades del mundo, todo esto conlleva a la imposibilidad de tener más vías o formas de hacer la modelación o descripción del software y así elevar el desarrollo en la industria del software cubano.

Como problema científico se tiene que en la UCI no se aprovechan al máximo los conocimientos que conforman la teoría relacionada con los métodos formales tanto en la documentación, como en la producción de software.

Dentro del amplio espectro que abordan las técnicas de modelación de información este trabajo profundiza en los métodos formales aplicados en la Ingeniería de Software.

El objetivo general de este trabajo es diseñar una estrategia de trabajo para aplicar los métodos formales en la producción de software en la UCI.

Asociado al mismo existen una serie de objetivos específicos:

1. Proponer un marco de trabajo para la aplicación de los métodos formales en los proyectos productivos que se desarrollan en la UCI.
2. Incorporar la enseñanza de los métodos formales en el proceso de aprendizaje de la Ingeniería de Software en la universidad

Como idea a defender se tiene que la aplicación de los métodos formales en los proyectos productivos que se desarrollan en la UCI, logrará una mayor calidad de los productos que se obtienen.

Para dar cumplimiento a los objetivos trazados se realizarán las tareas siguientes:

1. Investigar sobre los métodos formales y cómo se estudia en el resto del mundo, describiendo sus características, principios y fundamentos para conocer de que se tratan estas técnicas.
2. Buscar diferencias entre su plan de estudio en Cuba y en otras partes del mundo para poder vincularlo con las asignaturas del perfil.
3. Investigar sobre su aplicación en el desarrollo del software para utilizarlas en la producción de software en la UCI.
4. Buscar herramientas automatizadas para implementar el uso de los métodos formales.

El contenido de esta investigación está distribuido en dos capítulos, el primero es toda la fundamentación teórica, se explica el concepto de los métodos formales y todo lo referente a estas técnicas para el desarrollo de software, así como su enseñanza en el resto del mundo y herramientas para su implementación. En el segundo capítulo se hace un estudio de la relación de estos métodos con las demás asignaturas del perfil y se traza una estrategia para su inserción y aplicación en la Ingeniería de Software como combinación con los temas tratados en esta asignatura.

# Capítulo 1: Fundamentación Teórica

En este capítulo serán tratados algunos conceptos imprescindibles para la comprensión de los métodos formales en la Ingeniería de Software. Para ello, se parte de analizar que es la Ingeniería de Software, para luego llegar a conocer que son los Métodos Formales. También se abordará sobre la importancia de estos, su aplicación en la industria del software y algunos de los lenguajes de especificación que se basan en estos métodos. Además trata sobre su enseñanza en otros países del mundo y algunas herramientas para el trabajo con dichos métodos.

## 1.1-. Métodos formales en la Ingeniería de Software

La ingeniería del software es una disciplina que se ocupa de la búsqueda de métodos, técnicas y herramientas para mejorar la calidad del software de un sistema informático. (1) Una de las técnicas que se puede utilizar en el largo proceso de confección del software, si se quiere un producto terminado con mayor calidad y seguridad, son los métodos formales.

“Los métodos formales son técnicas de base matemática para describir las propiedades del sistema. Estos proporcionan marcos de referencia en el seno de los cuales las personas pueden especificar, desarrollar, verificar los sistemas de manera sistemática.” (2)

Otras definiciones los enmarcan en: “… técnicas que ayudan en la reducción de los errores introducidos en un sistema, particularmente en los primeros tiempos del diseño.” (3)

El método, notación o técnica formal que conviene emplear depende de la fase de desarrollo del sistema. En las fases iniciales del proyecto es preferible utilizar técnicas orientadas a propiedades, que permitan capturar los requisitos del sistema, estas técnicas son las más comunes. Sin embargo, al avanzar en el desarrollo se impone la necesidad de emplear técnicas constructivas, que permitan estructurar el sistema e identificar subcomponentes de cara a la implementación.

Suele denominarse técnicas de especificación formal a todo el proceso de definición de reglas, condiciones, variables, etc., y a los lenguajes cuyo vocabulario, sintaxis y semántica están formalmente definidos.

La esencia en el desarrollo de métodos formales es la creación de herramientas que deben combinar varias técnicas para obtener lo mejor de cada una en cada fase de la evolución del sistema y se caracterizan como la unión de un lenguaje formal y la inferencia mecánica. “El lenguaje formal se fundamenta en alguna lógica o cálculo con una sintaxis y semántica determinadas que permite expresar las propiedades de un dominio matemático de manera clara”(4), la amplitud de este dominio establece la capacidad expresiva del lenguaje. “La inferencia mecánica se refiere al sustento operacional. Cuando el lenguaje formal está fundamentado en una lógica, su inferencia mecánica se basa en un demostrador automático de teoremas. (4)

Lo métodos formales fuerzan al especificador a pensar con más cuidado sobre el problema en cuestión debido a su rigor matemático.

## 1.2-. Importancia de los Métodos Formales

En sistemas donde la seguridad y la integridad sea un factor primordial, un fallo puede pagarse muy caro, cuando esto ocurre se pueden tener graves consecuencias económicas o incluso puede ocasionar la pérdida de vidas humanas. En dichas situaciones es esencial descubrir los errores antes de poner en operación la computadora.

Los métodos formales reducen drásticamente los errores de especificación, y consecuentemente son la base del software que tiene pocos errores una vez que el cliente comienza a utilizarlo. Permiten al ingeniero del 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 y la construcción de sistemas que operen confiablemente a pesar de su complejidad.

Son una parte muy importante en el proceso de desarrollo de un software, ya que permite verificar la calidad del diseño de software mediante evaluaciones, comprobando si lo que se esta haciendo está bueno o malo.

### 1.3-. Aplicación de los Métodos Formales.

Los métodos formales se pueden aplicar en todas las fases del desarrollo del sistema. En la fase de requisitos para clarificar las exigencias del consumidor, en el diseño refinan los módulos, prueban la generación del caso en la validación y en la documentación sostiene y analiza la evolución.

Tradicionalmente, el uso de estos se toma como una fase separada entre la formulación del análisis y diseño y la implantación y verificación del sistema. Sin embargo, gracias a los modelos de cómputo abstracto que semánticamente pueden sustentarse, es posible que la aplicación de los métodos formales se establezca como una correspondencia partiendo del análisis y conduciendo a la implantación automática y confiable, refinando crecientemente e interactivamente el diseño conforme las necesidades así lo requieran.

Aún cuando la aplicación de métodos formales no garantiza a primera instancia que un sistema este hecho correctamente, facilita considerablemente el análisis de las propiedades de este, mostrando posibles inconsistencias, ambigüedades o propiedades incompletas que de otra forma pasarían desapercibidas.

Esta emergente tecnología encuentra sitio en prácticamente todo ámbito: desde sistemas operativos distribuidos, lenguajes de programación, dispositivos de hardware, etc. hasta aplicaciones de Inteligencia Artificial, como asistentes personales (interacción humano computadora), comercio electrónico, etc.

Las necesidades para asegurar interacción confiable en tales sistemas deben ser satisfechas, por lo que el desarrollo de métodos formales encuentra un campo fértil para proponer y desarrollar soluciones a la amplia gama de retos existentes.

Para aplicar los métodos formales el ingeniero del software debe tener conocimiento de la notación matemática asociada a los conjuntos y a las sucesiones, y a la notación lógica que se emplea en el cálculo de predicados. “Lo primero es definir la invariante de datos, el estado y las operaciones (ver epígrafe 1.4) para el funcionamiento de un sistema. Y luego la notación y la heurística asociados con los conjuntos y especificaciones constructivas, operadores de conjuntos, operadores lógicos y sucesiones, que forman la base de los métodos formales.” (2)

Cuando se aplican estos métodos se produce una especificación representada en un lenguaje por ejemplo: Z, VDM, etc. Pudiendo comprobar el resultado obtenido aplicando pruebas lógicas a cada función del sistema debido a que los métodos formales utilizan la matemática discreta como mecanismo de especificación para demostrar que una especificación es correcta.

- **Aplicación en la confección de tarjetas inteligentes.**

“Las tarjetas inteligentes son dispositivos diseñados para almacenar y procesar datos confidenciales, y pueden actuar como interfaces para proveer acceso seguro a las aplicaciones y servicios.” (5) Su uso se incrementa cada vez más por organizaciones comerciales y gubernamentales y se espera que cumplan un rol importante en asegurar pago electrónico confiable y confidencial.

Con el objetivo de que estas tarjetas ganen una amplia aceptación por el personal como dispositivos de computación confiables los productores, distribuidores y proveedores de aplicaciones para tarjetas inteligentes utilizan y son comúnmente aceptados los métodos formales, ya que esto sólo puede ser logrado mediante su utilización para diseñar, probar y verificar las plataformas y las aplicaciones de dichas tarjetas.

### 1.4-. Ventajas y desventajas de los Métodos Formales.

Tienen la ventaja de que los errores se pueden conocer más tempranamente, sin embargo para reconocerlos se debe manejar el contenido completo del proyecto.

Hay que invertir para la capacitación de profesionales que se enfrentan a los métodos formales lo que conlleva al interés de las empresas que sus trabajadores tengan una buena preparación, pero su costo es más elevado que el de los métodos tradicionales ya que se debe contar con colaboradores expertos, tanto en los métodos como en modelos matemáticos.

### 1.5-. Conceptos fundamentales implicados en la especificación matemática de los métodos formales.

Invariante de datos: “Es un conjunto de condiciones que son verdaderas durante la ejecución del sistema que contiene una colección de datos. Es una condición verdadera a lo largo de la ejecución del sistema que contiene una colección de datos. “(2)

Estado: “En los métodos formales, es un conjunto de datos almacenados a los que el sistema accede y es alterado por este.” (2)

Operación: “Es una acción que tiene lugar en un sistema y que lee o escribe datos en un estado.” (2)

“Una operación está asociada a dos condiciones: las precondiciones y las postcondiciones:

Una precondición define las circunstancias en que una operación en particular es válida.

Una postcondición define lo que ocurre cuando la operación ha finalizado su acción.” (2)

Esquema: “Estructuras parecidas a cuadros que presentan variables y que especifican la relación existente entre las variables.” (2)

## 1.6-. Lenguajes formales de especificación.

Un lenguaje formal es un conjunto de palabras (cadenas de caracteres) de longitud finita formadas a partir de un alfabeto (conjunto de caracteres) finito. El nombre lenguaje se justifica porque las estructuras que con este se forman tienen reglas de buena formación (gramática) e interpretación semántica (significado) en una forma muy similar a los lenguajes hablados. (6)

El método formal posee una base matemática estable que proporciona una forma de definir de manera precisa nociones tales como la consistencia y completitud, además de la especificación, la implementación y la corrección, dicha base normalmente vendrá dada por un lenguaje formal de especificación.

Un lenguaje de especificación formal suele estar compuesto de tres componentes primarios:

1. Una sintaxis que define la notación específica con la cual se representa la especificación. Usualmente se basa en una sintaxis derivada de la notación estándar de la teoría de conjuntos y del cálculo de predicados.
2. Una semántica que indica como representa el lenguaje los requisitos de un sistema. Se pueden usar abstracciones diferentes para describir el mismo sistema de formas distintas. La semántica de cada representación proporciona una visión complementaria del sistema.
3. Un conjunto de relaciones que definen las reglas que indican cuáles son los objetos que satisfacen correctamente la especificación.

El dominio sintáctico de un lenguaje de especificación formal suele estar basado en una sintaxis derivada de una notación estándar de la teoría de conjuntos y del cálculo de predicados.

El dominio semántico de un lenguaje de especificación indica la forma en que ese lenguaje representa los requisitos del sistema.

### 1.6.1-. Algunos tipos de lenguajes formales de especificación.

En la actualidad se utiliza toda una gama de lenguajes formales de especificación: CSP, LARCH, VDM y Z son lenguajes formales de especificación representativos. A continuación se describen algunos de los lenguajes de especificación más importantes que se basan en los métodos formales:

- **Lenguaje Z:**

"Z es una notación formal basada en la lógica de primer orden y en la teoría de conjuntos de Zermelo (de ahí la "Z") que utiliza los conceptos básicos de esa teoría (conjuntos, relaciones, funciones y variables) para describir sistemas y aplicaciones." (7)

Es el lenguaje de especificación matemática más usado por la industria, especialmente en sistemas de alta integridad, como parte del proceso del desarrollo del software y hardware principalmente en los países de Europa y los EE.UU. Ha experimentado la estandarización internacional debajo de ISO/IEC JTC1/2 WG19 en idiomas formales de la especificación.

El uso de este dio lugar a la concesión de una reina BRITÁNICA por el logro tecnológico en 1992 de su uso en el proyecto del CICS de IBM y contribuyó hacia una, en 1990 para que su uso especifique el estándar de IEEE para la aritmética Floating-Point binaria.

Está acompañado de una herramienta automatizada denominada "asistente de prueba", que almacena axiomas, reglas de inferencia y teoremas orientados a la aplicación que dan lugar a pruebas de corrección matemáticas de la especificación. Las especificaciones en Z se estructuran como un conjunto de esquemas.

En Z los esquemas sirven para modelar los aspectos tanto estáticos como dinámicos de un sistema. Define un cálculo para trabajar con esquemas mediante el cual es posible realizar numerosas operaciones con ellos: conjunción, disyunción, implicación, equivalencia, composición secuencial, etc. De esta forma podemos definir de forma totalmente independiente el esquema.

- **Object-Z o Z++**

Relacionada con la falta de modularidad de Z se encuentra otra de sus limitaciones: la falta de estructura en las especificaciones, lo que complica mucho aquellas de gran tamaño. En Z todas las variables son globales, y no es posible realizar especificaciones usando una estructura que no sea plana.

Una posible solución es introducir la orientación a objetos en Z, y para ello se han considerado fundamentalmente dos posibilidades:

1. Utilizar Z siguiendo un estilo orientado a objetos.

2. Extender Z para construir notaciones realmente orientadas a objetos (Object-Z o Z++).

Object-Z es muy similar a Z en detalle, sin embargo, difiere en lo que se refiere a la estructuración de los esquemas y a la inclusión de las funciones de herencia e instanciación. Trata de solucionar uno de los inconvenientes de Z, permitiendo agrupar un estado junto con un conjunto de operaciones sobre ese estado.

Incorpora también genericidad en sus clases, y la posibilidad de incluir invariantes de clase, es decir, predicados que se verifican a lo largo de la vida de los objetos de esa clase, y que deben ser respetados por las operaciones de la clase. Los invariantes se expresan como fórmulas en la lógica temporal.

- **Lenguaje VDM**

“VDM (Método del Desarrollo Viena) son técnicas para modelar sistemas de cálculo, Hacen un detallado análisis de los modelos, lo que conlleva un progresivo avance del diseño a la codificación.” (8)

Tiene sus orígenes en el trabajo del laboratorio de IBM Viena a mediados de los años setenta. Sin embargo, las notaciones formales y las herramientas para su uso se han convertido en un novedoso método y ahora se aplican en una amplia gama de sistemas.

- **Lenguaje CSP**

“CSP es un lenguaje diseñado para la programación concurrente de sistemas con memoria distribuida.”(9) Fue desarrollado por el grupo de investigación en programación de la Universidad de Oxford (Reino Unido). Su creador fue Hoare y fue presentado por primera vez en 1978. Fue uno de los desarrollos fundamentales en programación concurrente, muchos lenguajes reales se basan en este por ejemplo OCCAM, ADA, SR, etc.

Fue modificado como un lenguaje sin variables compartidas o direcciones comunes, es decir, asume que un programa está compuesto de n > 1 procesos secuenciales de comunicación. Está enfocado a desarrollar sistemas paralelos, el paralelismo es obtenido por la composición de un juego de procesos secuenciales que se ejecutan asincrónicamente tan pronto como ellos reciben todos sus datos de entrada provenientes de los mensajes. La salida es propagada para el siguiente proceso secuencial, mientras que no haya más mensajes para ser pasados, y el programa termine.

En este lenguaje la comunicación es por medio del paso de mensajes mediante una memoria compartida y es síncrona, su sintaxis y su semántica está enfocada a desarrollar sistemas paralelos con naturalidad, tiene mucha facilidad para implementar programas con múltiples entradas y salidas, especifica claramente el número de procesos concurrentes, variables, etc. Se dice que más que un lenguaje, es una filosofía de programación.

- **Lenguaje LARCH**

"Este lenguaje está pensado para la especificación exacta de los sistemas de cálculo. Permiten la especificación limpia de los programas de computadora y la formulación de pruebas sobre comportamiento del programa". (10)

Fue desarrollado sobre todo en los Estados Unidos en los años 80 y los años 90, implicando investigadores en Xerox, el MIT, y otros lugares. No se asemeja a la notación Z, es un

lenguaje para la especificación algebraica de los tipos de datos abstractos y una tiene una interfaz separada adaptable a los lenguajes que los programas deben ser escritos. En proyectos donde se trabaja con este lenguaje se desarrollan herramientas para apoyar al uso de especificaciones formales.

- **Lenguaje SLD**

"Es un lenguaje gráfico de especificación, estandarizado por la unión de telecomunicación internacional (ITU)". (11) Se ha estado desarrollando desde el año 80. Cada cuatro años se adopta una revisión actualizada del estándar del lenguaje. En 1992 fueron incluidos en SDL características orientadas a objetos. El estándar actual es SDL-2000, e introduce un número de nuevas características, incluyendo la dirección de excepción, un nuevo modelo de los datos, y estados compuestos. Aunque SDL es ampliamente utilizado en el campo de las telecomunicaciones, no se diseña específicamente para describir servicios de telecomunicaciones. Es utilizado en sistemas de comunicación y sistemas encajados. La notación gráfica, la semántica formal, y los conceptos orientados a objetos hacen a SDL un lenguaje de gran alcance y versátil para la especificación de sistemas y su puesta en práctica.

## 1.7-. Los diez mandamientos de los métodos formales

Bowen y Hinchley los hicieron como una guía para aquellos que los quieran empezar a utilizar. (2)

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

3. *Estimarás los costes. Los métodos formales tienen unos costes de arranques considerables. El entrenamiento del personal, la adquisición de herramientas de apoyo y la utilización de asesores bajo contrato dan lugar a unos costes elevados a la primera ocasión. Estos costes deben tenerse en cuenta cuando se esté considerando el beneficio obtenido frente a esa inversión asociada a los métodos formales.*
4. *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 los métodos formales por primera vez.*
5. *No abandonarás tus métodos formales de desarrollo. Es posible, y en muchos casos resulta deseable, integrar los métodos formales con los métodos convencionales y/o con 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.*
6. *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 natural a la especificación formal, para que sirva como mecanismo para reforzar la comprensión del sistema por parte de los lectores.*
7. *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.*
8. *No serás dogmático. El ingeniero de software debe reconocer que los métodos formales no son una garantía de corrección. Es posible que el sistema final, aún cuando se hayan desarrollado empleando métodos formales, siga conteniendo pequeñas omisiones, errores de menor importancia y otros atributos que no satisfagan nuestras expectativas.*

9. *Comprobarás, comprobarás y volverás a comprobar. Es muy importante la comprobación del software. Los métodos formales no absuelven al ingeniero del software de la necesidad de llevar a cabo unas comprobaciones exhaustivas y bien planeadas.*
10. *Reutilizarás cuanto puedas. A la larga la única forma racional de reducir los cortes 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.*

## 1.8-. Otros formalismos.

Otras notaciones de uso extendido entre la comunidad software que trabaja en la aplicación de métodos formales son: las máquinas de estado comunicantes, las redes de Petri, entre otros.

Las primeras “son notaciones que permiten describir diferentes máquinas de estado que se comunican mediante paso de mensajes.” (12) Usualmente también incluyen mecanismos para crear y destruir procesos, definir estructuras de datos, e incluir secuencias de control al estilo de los lenguajes imperativos. Los lenguajes más conocidos que siguen este tipo de notaciones son SDL [ITU-T, 1994], ESTELLE [ISO, 1989a] y PROMELA [Holzmann, 1991].

Las redes de Petri están basadas en las transiciones de estados para representar a los sistemas, aunque modela todo el sistema como un bloque en vez de representar los componentes como procesos. “Una red de Petri es una representación matemática de un sistema distribuido discreto.” (13) Las redes de Petri fueron definidas en los años 1960 por Carl Adam Petri. Son una generalización de la teoría de autómatas que permite expresar eventos concurrentes. Generalmente los problemas de flujo de trabajo se modelan con redes de Petri.

También esta la notación de máquina abstracta de B (B AMN) “es una notación formal orientada a objetos que ha sido utilizada con éxito en desarrollos industriales, sobre todo en Inglaterra” (12) (al igual que ocurre con Z). Actualmente se está intentando utilizar también para la especificación de sistemas reactivos y distribuidos, extendiéndola o combinándola con otros formalismos para aliviar sus puntos más débiles (no dispone de soporte semántico para la concurrencia, como le ocurre también a Z).

## 1.9-. Herramientas para aplicar los métodos formales.

- **Herramienta VERSA**

"Sistema de la ejecución, de la reescritura y de la verificación para el álgebra de comunicar los recursos compartidos (ACSR)". (14)

Para facilitar el ACSR en el diseño y el análisis de sistemas en tiempo real se desarrolló un sistema integrado de herramientas llamadas VERSA.

VERSA apoya tres tipos de técnicas del análisis (14):

1. Uso de reescribir reglas a los datos específicos del ACSR para deducir propiedades de sistema.

2. Construcción de una máquina del estado y de una exploración automática, y análisis del estado para verificar características de seguridad y para probar equivalencia de las formulaciones del proceso alternativo.

3. Ejecución interactiva de la especificación de proceso para explorar comportamientos específicos del sistema y para muestrear los rastros de la ejecución del sistema.

El sistema básico de VERSA tiene un interfaz para las descripciones del proceso de entrada, atarlas a los identificadores, y funcionamiento en ellos. VERSA amplía la sintaxis básica de ACSR con un número de convenciones de escritura prestadas de lenguajes de programación. Las especificaciones se pueden romper para arriba en componentes lógicos y recombinar usando la inclusión del archivo. Las constantes y las macros simbólicas se pueden definir

usando una notación del *#define* similar al lenguaje de programación de C. Se permiten los nombres variables de proceso puestos en un índice, los nombres del recurso, y las etiquetas del acontecimiento. Proporcionan los operadores puestos en un índice de la construcción de la composición y del sistema para funcionar en los sistemas de nombres puestos en un índice.

- **Herramienta DisCo**

La idea original de la herramienta es ayudar al usuario en validar una especificación. Cuando una especificación se ha extraído de un documento y se ha cargado a la herramienta se encuentran los errores que habrían sido difíciles de encontrar sin la herramienta.

"La herramienta de especificación DisCo consiste en tres porciones: interfaz utilizador del recopilador, el intérprete y el recopilador. Este último toma una especificación, realiza cheques semánticos y produce una representación interna ejecutable. La representación interna, se puede entonces ejecutar por el intérprete. El interfaz utilizador controla y visualiza la estructura estática de la especificación así como cambios dinámicos durante la ejecución". (15)

El usuario de la herramienta DisCo puede carecer del conocimiento y la motivación requerida para escribir “programas” para las animaciones, esta herramienta nos proporciona una forma de crear las exhibiciones tan fácilmente y automáticamente como sea posible. La herramienta DisCo proporciona las instalaciones con este propósito.

La aplicación de las especificaciones DisCo nos da una posibilidad de validar especificaciones probando. Está claro que las características críticas no pueden ser verificadas probando, porque la ejecución puede nunca alcanzar todos los estados posibles del sistema. Cuando se prueba una especificación, el usuario consigue otra vista de la especificación, que ayuda en entender el problema y ayuda en encontrar las características críticas que necesitan el razonamiento formal.

La visualización y la animación gráficas son útiles también cuando una especificación se demuestra a la gente. Por ejemplo, los expertos en el área de aplicación no son siempre capaces de entender pruebas formales, sino que después de ver la animación gráfica pueden

comentar respecto a la especificación, y sus preguntas pueden proporcionar la entrada valiosa para el razonamiento formal adicional de un informático.

- **Herramienta Cinderella SDL**

"Esta herramienta hace un modelado visual para desarrollar sistemas de software, servicios de las comunicaciones, protocolos, o cualquier clase de sistema basado en mensajes de señales. Cinderella SDL es basado en el lenguaje SDL". (16)

El analizador incremental de Cinderella SDL es el analizador más avanzado de SDL en el mercado hoy. Este analizador trabaja en el fondo mientras que estás desarrollando la especificación, incluso proporciona la corrección de error automática para ciertos tipos de errores. El explorador de la especificación es una herramienta configurable y extremadamente de gran alcance del usuario versátil para hojear y corregir una especificación. El simulador incorporado simula la especificación inmediatamente, es decir no hay compilación a un fichero ejecutable. Puedes incluso simular especificaciones parciales y poner al día la especificación mientras que simula.

## 1.10-. Conclusión del capítulo

1. La utilización de estas técnicas nos proporcionaría una mejor modelación del software, obteniéndose un producto final con una mayor calidad que los desarrollados con los métodos menos formales.
2. Si bien muchos creen que su difícil aprendizaje e implementación imposibilita grandemente su utilización en la producción de software, es cierto que su aplicación conllevaría a tener más seguridad en los sistemas y a una reducción considerable de los errores, que fácilmente se erradican en el flujo de trabajo de prueba.

# Capítulo 2: Inserción de los Métodos Formales en la carrera de Ingeniería Informática.

En este capítulo se hace un estudio de los temas de las asignaturas que se cursan en la carrera y que guardan relación con los conocimientos de los métodos formales, también se relacionan estos contenidos con los estudiados en la ingeniería de software y se propone vincular esta técnica menos formal con las técnicas formales y así lograr una mayor calidad en los software realizados en la UCI. Además, cuáles integrantes del equipo de desarrollo necesariamente deberán conocer estas técnicas para una mejor aplicación en los proyectos productivos y en cuáles de los proyectos que se han realizado en esta universidad se puede aplicar con mayor o menor énfasis dichos métodos.

## 2.1-. Vínculo de los Métodos Formales con las asignaturas del perfil de un Ingeniero Informático.

La enseñanza de los métodos formales está orientada a transmitir al alumno las posibilidades teóricas que ofrecen dichos métodos, a que desarrollen habilidades técnicas en el campo del diseño de la arquitectura, que analicen y adquieran destreza en el uso de estos métodos como sólida base matemática sobre la que construir procedimientos automatizables para el desarrollo de un sistema software.

Además también está orientada al estudio de la Lógica de Primer Orden, con sus propiedades y aplicaciones en diversas áreas en particular los métodos de demostración de teoremas y la interpretación de las fórmulas del lenguaje, tratándose así aspectos sintácticos y semánticos de los lenguajes, haciendo la introducción a la teoría de pruebas y a la teoría de modelos respectivamente.

El desarrollo de disciplinas y tecnologías como: la inteligencia artificial, el control automático, la psicología, la biología, la sociología, la economía, las finanzas y la administración de

negocios, necesitan de la de formalización del conocimiento impreciso o expresado exclusivamente de manera lingüística.

Por esta razón muchas de las asignaturas o disciplinas del perfil de un Ingeniero Informático se vinculan con los fundamentos de los métodos formales.

Para trabajar con métodos formales es necesario tener conocimiento de la Matemática Discreta principalmente sobre los conjuntos, las sucesiones y el cálculo de predicado.

También tienen una amplia relación con la Lógica Difusa estudiada en Inteligencia Artificial y se utilizan en aplicaciones de interacción humano-computadoras.

En las Ciencias Empresariales la elaboración y aplicación de métodos formales que reflejen el conocimiento empresarial, y su proyección estratégica, dan lugar a la más elevada forma de Inteligencia Empresarial.

Dentro de la formación de un ingeniero informático de la UCI existen toda una gama de asignaturas que el estudiante debe cursar a lo largo de su carrera; este esquema es una representación de todas estas asignaturas. Se dividen en tres departamentos fundamentales, el que incluye las materias de la especialidad, el de ciencias básicas y el de humanidades; incluidas en los departamentos de la especialidad y el de ciencias básicas se encuentran algunas asignaturas donde se imparten temas relacionados con los conocimientos de los métodos formales y que forman la base para el aprendizaje de los mismos, seguidamente se destacan cuáles son estas asignaturas y en cuáles se pretende que se profundicen dichos contenidos para un mejor desarrollo profesional del ingeniero:

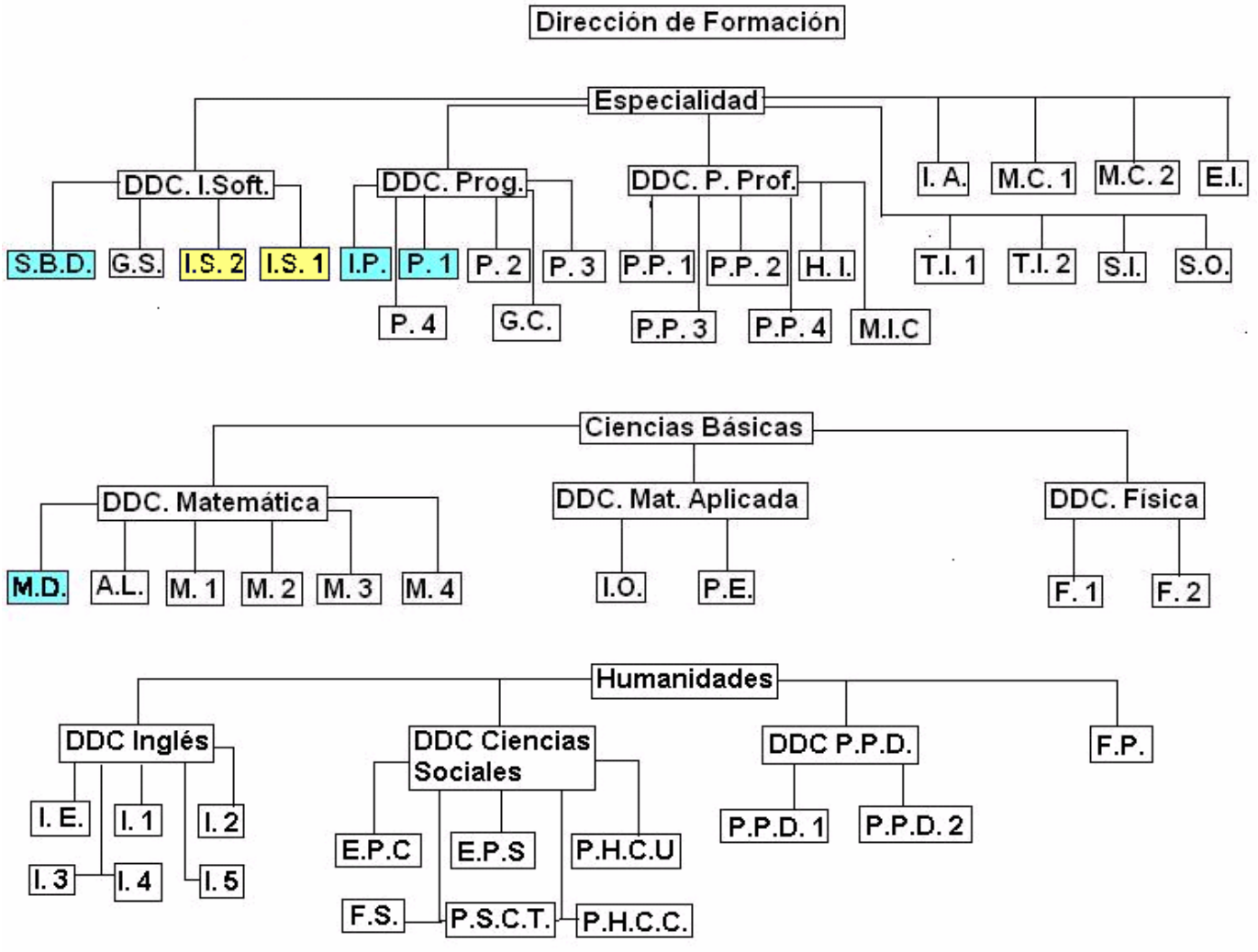



Figura 1 Representación de las asignaturas del perfil.

Leyenda:

**DDC. I. Soft:** Departamento Central de Ingeniería de Software.

**S.B.D:** Sistemas de Base de Datos.

**G.S:** Gestión de Software.

**I.S 2:** Ingeniería de Software 2.

**I.S 1:** Ingeniería de Software 1.

**DDC. Prog.:** Departamento Central de Programación.

**I.P:** Introducción a la Programación.

**P. 1:** Programación 1.

**P. 2:** Programación 2.

**P. 3:** Programación 3.

**P. 4:** Programación 4.

**G.C:** Gráfico por Computadora.

**DDC. P. Prof**.: Departamento Central de Práctica Profesional.

**P.P. 1:** Práctica Profesional 1.

**P.P. 2:** Práctica Profesional 2.

**H.I:** Historia de la Informática.

**P.P. 3:** Práctica Profesional 3.

**P.P. 4:** Práctica Profesional 4.

**M.I.C:** Metodología de la Investigación Científica.

**I.A:** Inteligencia Artificial.

**M.C. 1:** Máquina Computadora 1.

**M.C. 2:** Máquina Computadora 2.

**E.I.:** Ética Informática.

**T.I. 1:** Teleinformática 1.

**T.I. 2:** Teleinformática 2.

**S.I.:** Seguridad Informática.

**S.O.:** Sistema Operativo.

**DDC. Matemática:** Departamento Central de Matemática.

**M.D.:** Matemática Discreta.

**A.L.:** Álgebra Lineal.

**M. 1:** Matemática 1.

**M. 2:** Matemática 2.

**M. 3:** Matemática 3.

**M. 4:** Matemática 4.

**DDC. Matemática Aplicada:** Departamento Central de Matemática Aplicada.

**I.O.:** Investigación de Operaciones.

**P.E.:** Probabilidad y Estadística.

**DDC. Física:** Departamento Central de Física.

**F. 1:** Física 1.

**F. 2:** Física 2.

**DDC. Inglés:** Departamento Central de Inglés.

**I.E.:** Inglés Elemental.

**I. 1:** Inglés 1.

**I. 2:** Inglés 2.

**I. 3:** Inglés 3.

**I. 4:** Inglés 4.

**I. 5:** Inglés 5.

**DDC.Ciencias Sociales:** Departamento Central de Ciencias Sociales.

**E.P.C.:** Economía Política del Capitalismo.

**E.P.S.:** Economía Política del Socialismo.

**P.H.C.U.:** Panorama Histórico y Cultural Universal.

**F.S.:** Filosofía y Sociedad.

**P.S.C.T.:** Problemas Sociales de la Ciencia y la Tecnología.

**P.H.C.C.:** Panorama Histórico Cultural Cubano.

**DDC. P.P.D.:** Departamento Central de Preparación para la Defensa.

**P.P.D. 1:** Preparación para la Defensa 1.

**P.P.D. 2:** Preparación para la Defensa 2.

**F.P.:** Formación Pedagógica.

*Matemática Discreta*

Esta asignatura se cursa en el primer semestre del primer año de la carrera. Pertenece al departamento de Ciencias Básicas, aquí se estudian, entre otros, los temas: Teoría de Conjuntos, Operadores de Conjuntos, Lógica Proposicional y Notación lógica del cálculo de predicados, que forman los conocimientos básicos para aprender los métodos formales. Todos estos, se utilizan para desarrollar especificaciones constructivas, definir los estados, los invariantes de datos y las operaciones, así como las precondiciones y las postcondiciones en dichos métodos.

*Introducción a la Programación*

Esta materia se estudia en el primer semestre del primer año, esta incluida en el departamento de la Especialidad. En dicha asignatura se tratan conocimientos sobre los lenguajes de programación, muy importantes también a la hora de aplicar un lenguaje de especificación formal dentro de los métodos formales, por ejemplo: definición de clases y declaración de métodos y atributos, lo que ayuda considerablemente a definir los esquemas y las operaciones de dichos lenguajes de especificación.

*Programación 1*

Se cursa en el segundo semestre de primer año, pertenece al departamento de la Especialidad. En esta asignatura se comienza con la programación orientada a objetos, se estudian temas relacionados con la herencia de clases, que también se introducen en los lenguajes de especificación formal orientados a objetos de los métodos formales como Object Z o Z++.

*Sistemas de Base de Datos*

Esta asignatura se estudia en el segundo semestre del segundo año, pertenece al departamento de la Especialidad, aquí se estudian temas como el Álgebra y Cálculo de relacional, que no es más que una profundización del trabajo con conjuntos que como se explica anteriormente es la base para el aprendizaje de los métodos formales.

Todo esto se representa en la siguiente tabla:

| Año | Semestre | Asignatura | Temas | Relación con los Métodos Formales |
|---|---|---|---|---|
| 1 | 1 | Matemática Discreta | -Teoría de Conjuntos.<br>-Operadores de Conjuntos.<br><br>-Lógica Proposicional. | -Hacer especificaciones constructivas.<br>-Definición de estados, invariantes |

| | | | | |
|---|---|---|---|---|
| | | | -Notación lógica del cálculo de predicados | de datos y operaciones. |
| 1 | 1 | Introducción a la Programación | -Definición de clases.<br>-Declaración de métodos y atributos. | Definición de los esquemas y las operaciones en lenguajes de especificación formal. |
| 1 | 2 | Programación 1 | -Introducción a la programación orientada a objetos.<br>-Herencia de clases. | -Programación orientada a objetos en lenguajes de especificación.<br>-Definición de la herencia en estos lenguajes. |
| 2 | 2 | Sistemas de Base de Datos | -Introducción al Álgebra y Cálculo relacional | |

Tabla 1: Vínculo de los métodos formales con otras asignaturas.

## 2.2-. Combinación de los Métodos Formales con la Ingeniería de Software estudiada en la universidad.

Se pretende que los conocimientos que existen de los métodos formales se combinen con las técnicas menos formales que estudiamos en la carrera para así lograr un resultado mejor que el obtenido hasta el momento. A continuación se hará una descripción por cada flujo de trabajo de cuáles son las técnicas que se podrían incluir o profundizar en el estudio de las asignaturas Ingeniería de Software 1 y 2 en la UCI para una mejor realización de un producto de software.

- **Modelamiento del Negocio.**

En el flujo de trabajo de *Modelamiento del Negocio* se definen las reglas del negocio, que en los métodos formales serían las invariantes de datos y se formulan de forma matemática, con la utilización de las operaciones y la teoría de conjuntos, permitiendo una mayor exactitud en estas restricciones. Para una mejor comprensión este aspecto definiremos el invariante de datos mediante un ejemplo:

## Ejemplo 1: Un gestor de bloques. (2)

Una de las partes más importantes de los sistemas operativos es el subsistema que mantiene archivos que hayan sido creados por usuarios. Una parte del subsistema de archivos es el gestor de bloques. Los archivos del almacén de archivos están formados por bloques de espacio que se almacenan en algún dispositivo de almacenamiento de archivos. Durante el funcionamiento de la computadora los archivos van siendo creados y borrados, lo cual requiere la adquisición y liberación de bloques de almacenamiento. Para abordar este bloque, el subsistema de archivo mantendrá una reserva de bloques libres y seguirá también la pista de aquellos bloques que se estén utilizando en el momento. Cuando se liberen bloques procedentes de un archivo borrado lo normal será añadirlos a una cola de bloques que están a la espera de ser añadidos al depósito de bloques que no se utilizan.

En la Figura 2 se muestra un cierto número de componentes: la reserva de bloques no utilizados, los bloques que en la actualidad forman parte de los archivos administrados por el sistema operativo y aquellos bloques que estén esperando para ser añadidos a la reserva de espacio. Los bloques que están a la espera se mantienen en una cola y cada elemento de la cola contiene un conjunto de bloques procedentes de algún archivo borrado.

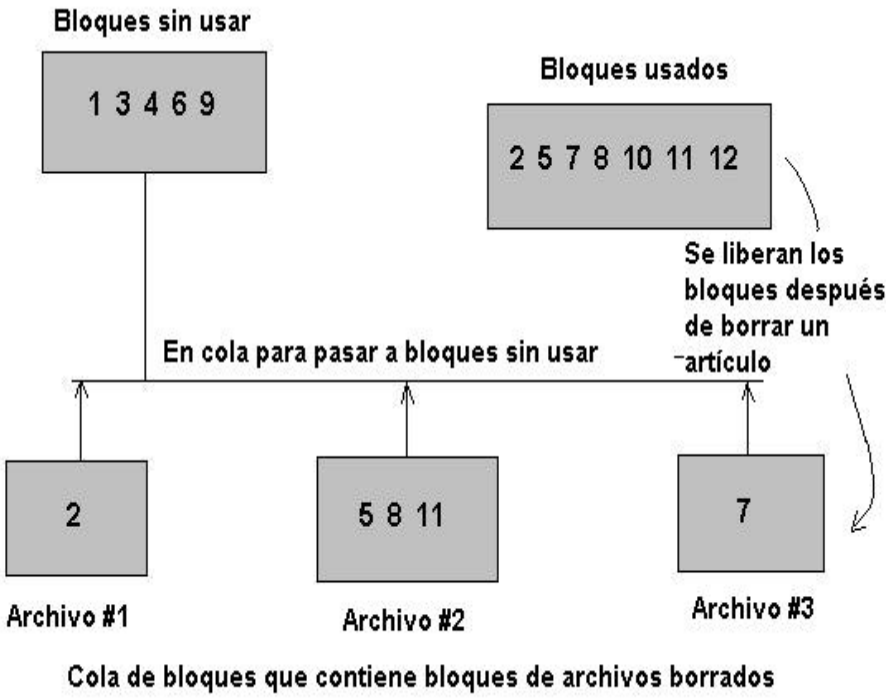



Figura 2: Un gestor de bloques.

El invariante de datos para este ejemplo expresado en lenguaje natural es:

-No habrá ningún bloque que esté a la vez usado y sin usar.

-Todos los conjuntos de bloques almacenados en la cola serán subconjuntos de la colección de bloques utilizados en ese momento.

-No existirán elementos de la cola que contengan los mismos números de bloque.

-La colección de bloques usados y bloques sin usar será la colección total de bloques de que consten los archivos.

-En la colección de bloques sin usar no existirán números de bloques duplicados.

-En la colección de bloques usados no existirán números de bloques duplicados.

Utilizando la notación matemática las reglas se escribirían como invariante de datos de la siguiente manera:

$$
\begin{aligned}
&\text{usados} \cap \text{libres} = \emptyset \wedge \\
&\text{usados} \cup \text{libres} = \text{TodosBloques} \wedge \\
&\forall i : \text{dom ColaBloques} \,.\, \text{ColaBloques } i \subseteq \text{usados} \wedge \\
&\forall i,j : \text{dom ColaBloques} \,.\, i \neq j \Rightarrow \text{ColaBloques } i \cap \text{ColaBloques } j = \emptyset
\end{aligned}
$$

Figura 3: Un invariante de datos.

- **Requerimientos**

En el flujo de trabajo de *Requerimientos* se hace el levantamiento de requisitos, en esta fase es más importante la especificación flexible de un sistema en función de sus propiedades más importantes, la descripción resultante de esta fase constituye la arquitectura inicial del sistema.

En esta parte del proceso de desarrollo es donde aparecen la mayor cantidad de funcionalidades no requeridas por el usuario final, que pueden rectificarse o permanecer en gran parte del proceso, debido a que todas se formulan con el lenguaje natural y cotidiano, entonces aparecen toda una serie de problemas como son las contradicciones, ambigüedades, vaguedad, etc. Con la aparición de los métodos formales y con ello, la utilización de la teoría de conjuntos y notaciones lógicas para crear sentencias claras de hechos o requisitos, se erradican, en esta etapa, toda esta gama de problemas, ya que permiten también la abstracción y se definen sentencias o especificaciones funcionales que no permiten la ocurrencia de estos problemas, pues describen de forma exacta una situación física o el resultado de una acción, la vaguedad desaparece completamente.

Se especifican las propiedades escribiendo un predicado por cada propiedad y se hace más énfasis en los requisitos de seguridad. Las restricciones se describen mediante la definición de estados y operaciones.

Siguiendo el Ejemplo 1 del gestor de bloques (2), el estado se describe como:

*Usados, libres:* P *BLOQUES*

*ColaBloques: seq* P *BLOQUES*

Esta descripción se asemeja a la declaración de variables de un programa. Afirma que *usados* y *libres* serán los conjuntos de bloques y que *ColaBloques* será una sucesión (seq) donde cada elemento es un conjunto de bloques.

Se definen dos operaciones, la primera es la que elimina un elemento de la parte anterior de la cola de bloques, la precondición es que debe existir al menos un elemento en la cola para que esto pueda ocurrir, matemáticamente se escribe: *#ColaBloques > 0,* la postcondición es que es preciso eliminar la cabeza de la cola y ubicarla en la colección de bloques libres, y ajustar la cola para que muestre la eliminación, todo esto se hace de la siguiente manera:

*usados´= usados*\cabeza *ColaBolques* ^

*libres´= libres* U cabeza *ColaBloques* ^

*ColaBloques´= cola ColaBloques*

*(Una convención que se utiliza en muchos métodos formales es que el valor de una variable después de una cierta operación lleva el signo de prima).*

La segunda operación es la que se encarga de añadir una cola de bloques *BloquesA* a la cola de bloques. La precondición es que *BloquesA* sea en ese momento un conjunto de bloques usados y se expresa así: *BloquesA usados.*

La postcondición es que el conjunto de bloques se añade al final de la cola de bloques y el conjunto de bloques libres y usados permanece invariable:

$$ColaBloques' = ColaBloques \frown \langle ABlocks \rangle \land$$
$$usados' = usados \land$$
$$libres' = libres$$

Figura 4: Postcondición de una operación.

- **Análisis**

En el flujo de trabajo de *Análisis* se traducen los requisitos a una especificación que describe cómo implementar el sistema.

Con los métodos formales se logran utilizar distintas abstracciones semánticas para describir un mismo sistema de diferentes maneras.

- **Diseño**

En el flujo de trabajo de *Diseño* es necesario poder describir los componentes del sistema, su interfaz y cómo interactúan entre ellos. Es, por tanto, necesario poder expresar la estructura. En esta fase también se pueden aplicar estos métodos pues la especificación del diseño del sistema muestra las propiedades del modelo.

Un procedimiento que se utiliza con mucha frecuencia en el diseño con métodos formales consiste en la construcción del sistema mediante refinamientos sucesivos. El proceso comienza con una descripción muy abstracta (una arquitectura inicial, generalmente carente de detalles que condicionen su implementación) y se construye de forma incremental, como una secuencia de refinamientos hasta alcanzar el producto final, que satisface todos los requisitos del usuario. En cada refinamiento, se toma como entrada el diseño resultante de la

etapa anterior y un objetivo (un nuevo requisito de usuario, el aumento del nivel de detalle de la descripción, etc.). Se toma una decisión de diseño, se realiza una transformación y se obtiene el nuevo sistema que cumple el objetivo.

Para la definición de clases del diseño, se definen variables como se hace en un lenguaje de programación por lo que resulta considerablemente más fácil hacer la transición de un modelo del diseño a la implementación, además se empieza el razonamiento del sistema desde una fase más temprana y se comienza a pensar en el código mucho antes de llegar a la fase de implementación donde los defectos ya están tan asentados que resulta de un gran coste económico su corrección.

- **Implementación**

En el flujo de trabajo de *Implementación* también se aplican en gran medida los métodos formales ya que con un lenguaje de especificación formal se podría expresar mucho más de lo que se representa con un lenguaje de programación. A continuación se explica con más profundidad lo expresado anteriormente.

Cuando se disponga de una especificación del sistema que recoja satisfactoriamente todos los requisitos del usuario y cuente con el suficiente nivel de detalle, hay que proceder a su implementación en la tecnología elegida.

Todo código de un programa es ciertamente una notación matemática algo verbosa, pero el lenguaje de programación que normalmente se utiliza en el flujo de trabajo de Implementación no es un buen lenguaje de especificación porque solamente pueden representar funciones computables.

El proceso de traducción de un lenguaje de especificación a un lenguaje de implementación se puede automatizar en gran medida. Por ello, esta fase se suele decir que es semiautomática. Por la misma razón, resulta preferible realizar las tareas de mantenimiento a nivel de lenguaje de especificación.

Los lenguajes de especificación formal son un medio excelente para el modelado y se utilizan en dependencia de las características del sistema a desarrollar por ejemplo si es un sistema con orientación a objetos se utiliza el Object Z (ver epígrafe 1.6), para la programación de sistemas concurrentes se usa el CPS (ver epígrafe 1.6), etc. Además se pueden emplear diferentes lenguajes en un mismo formalismo, cada uno capaz de modelar con mayor fidelidad un aspecto distinto del sistema. Tienen una semántica más amplia que hace posible especificar el comportamiento del sistema, deben ser capaces de expresar ideas como:

-Para todo X de un conjunto infinito A existe un Y de un conjunto infinito B tal que la propiedad P es válida para X e Y.

También como en otros lenguajes de programación se puede reutilizar código. Una subrutina o procedimiento en un lenguaje de programación es análogo a un esquema en un lenguaje de especificación formal.

Para un mejor entendimiento de la aplicación de un lenguaje de especificación en esta fase se explicará brevemente como es la definición de un esquema mediante un ejemplo.

Retomando el ejemplo 2 un esquema en el lenguaje Z se define como: (2)

*Gestionar Bloques*

| |
|---|
| *Usados, libres: (P conjunto potencia) BLOQUES*<br>*ColaBloques: seq (P) BLOQUES* |

usados ∩ libres = ∅ ∧
usados ∪ libres=TodosBloques ∧
∀ i : dom ColaBloques . ColaBloques i ⊆ usados ∧
∀ i,j :dom ColaBloques . i ≠ j ⇒ColaBloques i ∩ ColaBloques j = ∅

Figura 5: Representación de un esquema.

Se compone de dos partes, la primera, por encima de la línea central, representa las variables del estado, mientras que la que se encuentra por debajo de la línea central describe el invariante de datos. También se pueden utilizar para describir operaciones, por ejemplo en las siguientes operaciones se describe cómo eliminar un elemento de una cola de bloques y cómo se añade una colección de bloques al final de la cola respectivamente:

*(Como se utiliza el esquema anterior en esta operación se representa mediante ∆GestorBloques, y su inclusión da como resultado todas las variables que componen el estado disponible en el esquema EliminarBloques y asegura que el invariante de datos se mantendrá antes y después que se ejecute la operación)*

*Eliminar Bloques*

| ∆*GestorBloques* |
| --- |
| *#ColaBloques > 0*<br>*usados´= usados*\cabeza *ColaBolques* ^<br>*libres´= libres* U cabeza *ColaBloques* ^<br><br>*ColaBloques´= cola ColaBloques* |

*Añadir Bloques*

| ∆*GestorBloques*<br>*BloquesA?: BLOQUES* |
| --- |

*BloquesA?*⊆ usados

ColaBloques' = ColaBloques ^ ⟨ABlocks⟩ ∧
usados' = usados ∧
libres' = libres

Figura 6: Representación de las operaciones Eliminar y Añadir Bloques.

*(Por convención en Z se le agrega el signo de interrogación (?), cuando es un elemento nuevo que se agrega a la cola, es decir una variable que no pertenece al estado)*

También como en los lenguajes de programación existen lenguajes de especificación formal orientados a objetos como es el caso del Object Z o Z++, que difiere de los demás en cuanto a la instanciación de una clase y en el que se incluye la función de herencia. A continuación y para clarificar esta situación se muestra un ejemplo de especificación en Object Z, se representa la especificación de una clase que describe una cola genérica que puede tener objetos de cualquier tipo (2).

*Cola[T]*

| |
|---|
| *numObjetosMax: N* |
| *numObjetosMax <=100*<br>*cola: seqT* |
| *#cola <= numObjetosMax* |
| INIT<br>*cola*= <> |

*Añadir*

| |
|---|
| Δ *(numObjetosMax)*<br>*elemento?: T* |

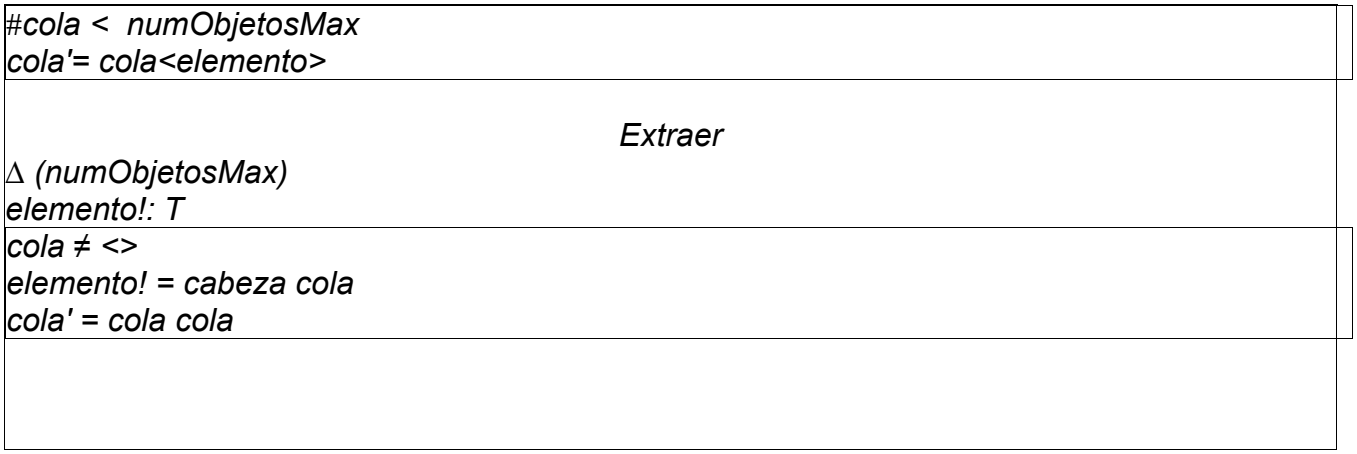



Figura 7: Representación de una clase.

Aquí se ha definido una clase que tiene una variable de instancia *cola,* con objetos del tipo T, donde T puede ser un entero, una factura, un grupo de elementos de configuración o bloques de memoria. Debajo de la definición están los esquemas que definen la clase. El primero define una constante que tendrá un valor no mayor de 100 y especifica la longitud máxima de la cola. Los 2 esquemas que siguen *Añadir* y *Extraer* definen los procesos de añadir y extraer un elemento de la cola. *(Se le agrega el signo de exclamación (!), cuando se refiere a un elemento que ya existe en la cola).*

Si quisiéramos utilizar un objeto definido por dicha clase sería relativamente simple de hacer. Por ejemplo supongamos que se está definiendo parte de un sistema operativo que manipulaba procesos, y que los procesos tenían que estar en dos colas, una con los procesos de alta prioridad y otra con aquellos de baja prioridad, la prioridad la utilice el planificador del sistema operativo con el propósito de decidir cuál es el siguiente proceso a ejecutar. La primera sentencia de Object Z necesaria instanciará la clase de cola general en una que contenga procesos. (2)

*ColaProc ==Cola [PROCESOS]*

*[procQ/ cola, proc?/ elemento?,  proc!/ elemento!]*

Se puede decir que todo lo que se hace es reemplazar el tipo *T* por *PROCESOS* en la definición de la clase, la cola general *cola* por la cola de procesos *procQ* y los parámetros generales de entrada y salida *elemento?* y *elemento!,* por aquellos parámetros de proceso *proc?* y *proc!*. (El símbolo / representa una forma de sustitución de texto.)

El siguiente paso es definir una clase que describe las dos colas. Esta utiliza las operaciones definidas en la clase *Cola.* Supongamos que las siguientes operaciones son necesarias:

- *añadirAlta*, añade un proceso a la cola de los procesos de alta prioridad.
- *añadirBaja*, añade un proceso a la cola de los procesos de baja prioridad.
- *INIT*, inicializa las colas de alta y baja prioridad.
- *totalProcesos* devuelve el número total de procesos que están en las colas.

*ParejaCola*

alta, baja: ColaProc

alta ≠ baja

INIT

alta.INIT ^ baja.INIT

*añadirAlta* Δ *alta.Añadir*

=

*añadirBaja* Δ *baja.Añadir*

=

TotalProcesos

total!: N

total! =: #alta + #baja

Figura 8: Representación de la herencia de la clase anterior.

La primera línea define el esquema, la segunda y tercera definen el estado y el invariante del estado. En este caso el estado se compone de dos colas de proceso bien diferenciadas: *alta* y *baja.*

A la definición del estado le sigue la definición de cuatro operaciones. *INIT* se define como la aplicación de la operación heredada *INIT* en las colas *alta* y *baja*; *añadirAlta* se define como la aplicación de la operación heredada *añadir* sobre el componente de estado *alta*; *añadirBaja* se definen como la aplicación de la operación heredada *añadir* sobre el componente de estado *baja*; y finalmente, la operación *TotalProcesos* se define como la suma de tamaños de las colas individuales *alta* y *baja*.

Consecuentemente esto es un ejemplo de instanciación en acción, donde los objetos definidos por un esquema de Object Z se utilizan en otro esquema. Con esto se puede definir la herencia.

- **Prueba**

Ahora se analizará el flujo de trabajo de *Prueba* en el que también se pueden utilizar los métodos formales y donde estos cuentan con una gran importancia.

Con su utilización en esta fase las especificaciones se pueden verificar matemáticamente para descubrir contradicciones y especificaciones no completas. Verifican el modelo probando que el modelo descrito verifica las propiedades del sistema.

El proceso de verificación implica la comparación de dos objetos formales: dos especificaciones del sistema, una especificación y una implementación, una especificación y una propiedad, etc. En cualquier caso, el paso inicial consiste en formalizar los objetos a comparar, es decir, crear modelos matemáticos que describan las propiedades de los objetos reales a los cuales representan.

¿Cómo se verifica la especificación?
Si partiendo de un estado donde el predicado es cierto la operación deja al sistema en un estado donde el predicado también es cierto.

En la actualidad, existen dos mecanismos más difundidos para la verificación formal de sistemas y que se pudieran utilizar para la verificación de los sistemas desarrollados en la universidad al aplicar en ellos los métodos formales, estos son: los demostradores de teoremas y el model checking (chequeo de modelo).

¿Cómo se hace la verificación a través de los demostradores de teoremas?

Se parte de que como se explicó anteriormente las propiedades del sistema se expresan mediante el empleo de alguna lógica matemática, que forma parte de un sistema formal donde se define un conjunto de axiomas y de reglas de inferencia. Una propiedad se define como un teorema, cuya veracidad es el objetivo a demostrar, con este objetivo se hace un razonamiento deductivo a partir de los axiomas y la aplicación de las reglas de inferencia, las definiciones y teoremas intermedios derivados a lo largo del proceso de demostración.

¿Cómo se realiza la verificación utilizando los model checking?

Se utilizan principalmente para sistemas de tiempo real. Primeramente se construye un modelo finito del sistema y se comprueba que una propiedad dada se satisface en dicho modelo.

Básicamente, la demostración se ejecuta mediante una búsqueda exhaustiva en el espacio de estados del modelo del sistema, por lo que para que termine, este espacio de estados deberá ser necesariamente finito.

Para detectar errores en las etapas de diseño e implementación también podrían utilizarse métodos como simulación guiada, verificación deductiva.

- En la simulación guiada se realizan experimentos con los sistemas antes de su implantación, específicamente se hace sobre una abstracción o modelo del sistema.
- En la verificación deductiva se utilizan reglas para demostrar matemáticamente la corrección de los sistemas, se puede utilizar en sistemas con un número infinito de estados.

Además, la verificación muestra que algunos códigos de programas son el reflejo exacto de un diseño utilizando una demostración matemática.

No requiere de esfuerzo para la validación inicial y gran parte de la comprobación del sistema tiene lugar durante la prueba de aceptación del sistema.

- **Planificación del Proyecto.**

En el flujo de trabajo de *Planificación del Proyecto* se debe estimar el costo de la aplicación de los métodos formales pues son considerables al iniciar un proyecto de software donde se utilicen, incluye el entrenamiento del personal, la adquisición de herramientas de apoyo y si se realiza el contrato a alguien para la asesoría. A la hora de considerar el beneficio obtenido con el producto también han de tenerse en cuenta.

- **Documentación.**

La Documentación en los métodos formales es muy importante, debe documentarse lo suficiente. Esta documentación sostiene y analiza la evolución del producto, y se recomienda que se hagan aclaraciones comentarios en lenguaje natural sobre todas las cosas para una mejor comprensión de todo el personal que lo lea.

## 2.3-. Descripción del Sitio Web.

Para profundizar el estudio de los métodos formales e incentivar a quienes quieran desarrollar sistemas utilizando estas técnicas formales se ha realizado como complemento de este trabajo de diploma un sitio Web, en donde se exponen algunos temas y se explica con mayor detalle los contenidos de esta materia.

El sitio cuenta con una página principal (Ver Anexo 1) donde se incita al lector a trabajar con lo métodos formales de la ingeniería de software, que conozca cuáles son las ventajas que ofrece en el desarrollo de software para la producción de un producto con una mayor calidad, y con un número reducido de errores en su elaboración. En fin, es una introducción y una motivación para el uso de estas técnicas por los ingenieros informáticos. En esta página existen varios botones que son vínculos a otras páginas del sitio, como son: Conceptos, Lenguajes, Mandamientos, Definiciones, Herramientas y Enseñanza.

En la página de *Concepto* (Ver Anexo 2) se exponen algunas definiciones de métodos formales para que el lector conozca formalmente cuál es el concepto de métodos formales.

En *Lenguajes* (Ver Anexo 3) se aborda sobre cuáles son las características de los lenguajes de especificación formal y sus ventajas sobre los lenguajes de programación. También se referencian algunos de los tipos de lenguajes de especificación que existen en la actualidad, con sus características específicas.

En la página de *Mandamientos* (Ver Anexo 4) se trata sobre los diez mandamientos de los métodos formales que hicieron Bowen y Hinchley, como una guía para todos los que es decidan a trabajar con los estos métodos.

En *Definiciones* (Ver Anexo 5) se abordan los principales conceptos implicados en las especificaciones matemáticas de los métodos formales y que forman una parte importante y básica del conocimiento de estos.

En *Herramientas* (Ver Anexo 6) se describen algunas de las herramientas fundamentales para el trabajo con los métodos formales, cuáles son sus características y específicamente para que se utilizan para el desarrollo de sistemas basados en estos métodos.

En la página *Enseñanza* (Ver Anexo 7) se hace una introducción a la enseñanza de los métodos formales, tratando de que el lector conozca sin mucho nivel de profundización, los primeros pasos para la aplicación de estos métodos, además las notaciones de algunos de los lenguajes de especificación formal, así como la definición de sus clases, realización de diferentes operaciones, etc.

## 2.4-. Integrantes del equipo de desarrollo que deben profundizar sus conocimientos sobre los Métodos Formales.

Para el desarrollo de un producto de software interviene un equipo de realización especializado en la creación de este tipo de proyecto. Comúnmente está formado por un líder del proyecto que dirige y controla toda la operación y por otra serie de personas especializadas en alguna de las etapas del proceso en particular.

En el flujo de trabajo de Modelamiento del Negocio se encuentran el *Analista de procesos del*

*negocio*, el *Diseñador del negocio* y el *Revisor del modelo del negocio*, que también se le llama *Arquitecto*. En el flujo de trabajo de Requerimientos está el *Analista del sistema*, el *Especificador del sistema*, el *Arquitecto* y el *Diseñador de interfaz usuario*. En el flujo de trabajo de Análisis y Diseño participan el *Arquitecto*, el *Ingeniero de casos de uso* y el *Ingeniero de componentes*. En la Implementación están el *Arquitecto*, el *Integrador*, el *Programador* y el *Ingeniero de componentes*. En el flujo de trabajo de Prueba participan el *Administrador de prueba*, el *Analista de prueba*, el *Diseñador de prueba* y el *Probador*.

En la siguiente tabla se representa la distribución completa de este equipo de desarrollo y se destacan cuáles son los trabajadores que deben tener una mayor preparación sobre los conocimientos de los métodos formales.

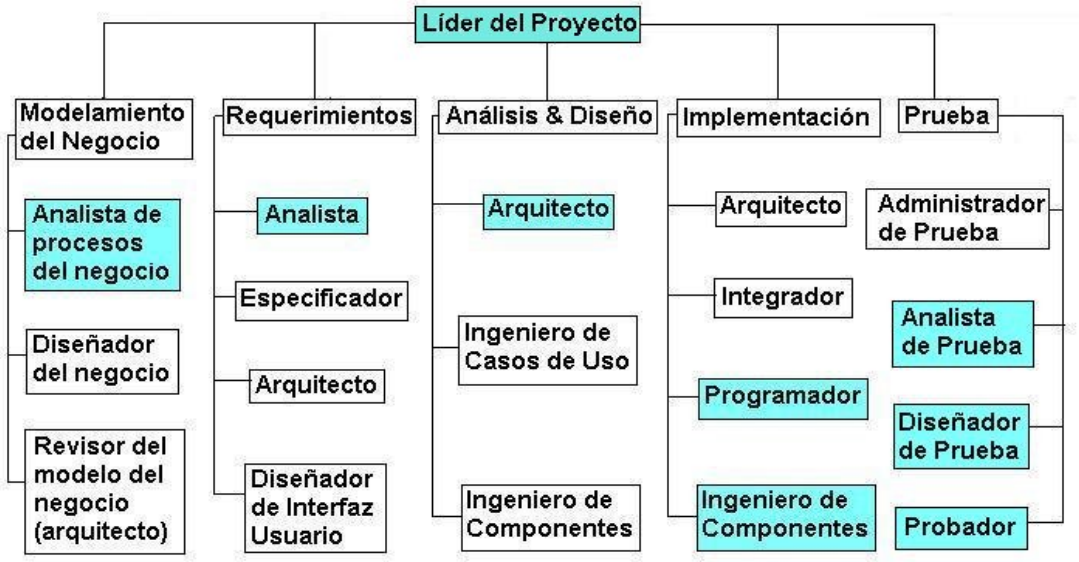



Figura 9: Representación del los integrantes del equipo de desarrollo.

El *líder del proyecto* debe tener un amplio conocimiento de los métodos formales como máximo responsable del proyecto, además de ser el encargado de estimar costes, definir los

requisitos que deben cumplir los integrantes del equipo, forma las prioridades del proyecto y controla el avance del proyecto durante toda la etapa de su desarrollo.

Dentro del flujo de trabajo de Modelamiento del Negocio se encuentra el *Analista de procesos del negocio*, este es el encargado de identificar las reglas del negocio, que se pueden modelar fácilmente y en un contexto más detallado y completo como son los métodos formales. También es el encargado de realizar en documento Visión y todo el que realice algo de la documentación del proyecto debe conocer el tema para que realice aclaraciones y/o comentarios necesarios para que otras personas comprendan con una mayor claridad las especificaciones matemáticas hechas en esta parte de desarrollo del proyecto.

Dentro del flujo de trabajo de requerimientos está el *Analista del Sistema*. Este es el encargado de realizar la captura de requisitos, actividad fundamental y de mayor importancia en la aplicación del los métodos formales, pues aquí es donde se encuentran los principales problemas en las técnicas menos formales y donde aparecen la ambigüedad, la vaguedad y los requisitos incompletos, introduciéndose aquí la mayor parte de los errores y funciones no deseadas por los usuarios finales del sistema. Todos estos problemas se erradican con facilidad si en esta etapa se utilizan los métodos formales. También se encarga de documentar el Plan de Gestión de Requisitos y de completar el Documento Visión.

En el flujo de Análisis y Diseño está el *Arquitecto*, este es el encargado de definir las clases del diseño y realizar el modelo del diseño, también es el que hace la descripción de la arquitectura por tanto debe conocer los métodos formales porque todas estas técnicas matemáticas son un medio excelente para el modelado y además sería muy cómodo y ventajoso la implementación de un producto a partir de un modelo del diseño realizado con los métodos formales.

El *Programador* se encuentra en el flujo de trabajo de Implementación este se encarga de implementar todo lo que resulta del flujo de trabajo de Diseño en un lenguaje de programación previamente establecido, como se explica en el epígrafe 1.6 del capítulo 1 para la aplicación de los métodos formales existe una serie de lenguajes de especificación formal que sirven para la implementación de los programas en dependencia de que sistema se quiera confeccionar, por tanto este integrante del equipo debe tener un amplio conocimiento de

dichos lenguajes de especificación y por consiguiente de los métodos formales para poder desarrollar el producto satisfactoriamente.

En este mismo flujo de trabajo está el *Ingeniero de Componentes*, este desarrollador es el que garantiza que cada componente implementado en este flujo de trabajo ya sea por un lenguaje de programación o por un lenguaje de especificación formal esté realizando la funcionalidad correcta.

Luego se encuentra el flujo de trabajo de Prueba, aquí se encuentra entre otros integrantes el *Analista de Prueba*, este es el que identifica y define las pruebas que requiere el proyecto, monitorea el progreso de la prueba y el resultado de cada ciclo de prueba, también se encarga de evaluar la calidad total del producto como resultado de las actividades de prueba. Cuando se aplican los métodos formales este flujo es de gran importancia pues las pruebas se van realizando a todo lo largo del proceso de realización del producto y no recae todo el peso en esta etapa, por eso es muy importante que se identifiquen con efectividad las pruebas que van requiriendo y también para evaluar la calidad final del producto con la aplicación de dichas técnicas.

En este mismo flujo está el *Diseñador de Prueba*, este define cuál método de prueba es más factible e identifica que técnicas y herramientas utilizar en las diferentes etapas de desarrollo, para saber que técnica utilizar cuando se aplican los métodos formales es necesario conocer que métodos se utilizan para detectar errores y cómo es que funcionan los modelos e implementaciones realizadas en los procesos anteriores.

El *Probador* también se encuentra en el flujo de trabajo de Prueba, este es el responsable por las pruebas realizadas, las conduce y registra sus resultados, por lo tanto debe conocer y seguir de cerca todo lo que se realice en este flujo de trabajo, y como se van detectando y arreglando los errores para documentar toda esta evolución.

## 2.5-. Aplicación de los Métodos Formales en los proyectos productivos que se desarrollan en la universidad.

Para la posterior aplicación de los métodos formales en los proyectos productivos que se están desarrollando actualmente en la universidad se ha hecho un estudio en una selección de los mismos, en diferentes facultades y en una escala de Mucho, Poco, Nada o Parcialmente, se exponen en qué medida se podrían aplicar las técnicas formales en la producción de la UCI, con el objetivo de lograr una mayor calidad de los productos finales.

| **Facultad** | **Nombre del proyecto** | **Descripción del proyecto** | **Utilización de los Métodos Formales.** | | | |
|---|---|---|---|---|---|---|
| | | | Mucho | Poco | Nada | Parcialmente |
| 10 | Biblioteca Nacional José Martí | Diseñar un sistema para la automatización de la Biblioteca Nacional. | X | | | |
| 10 | Portales | Desarrollo de sitios Web de interés de la universidad y del país. | | X | | |
| 10 | Intranet 2 | Desarrollo de la intranet de la UCI en Software Libre | | X | | |
| 4 | ERP MINFAR | Desarrollo de un ERP para el MINFAR | X | | | |
| 4 | ERP Cubano | Diseñar e implementar un sistema integral de gestión empresarial para las entidades presupuestadas y empresariales de Cuba. | X | | | |
| 4 | Tele Banca | Desarrollar la aplicación que tiene como objetivo fundamental gestionar el pago de servicio; brindar a las operadoras la información necesaria para responder a las preguntas más comunes que realizan los clientes | | | | X |
| 4 | Prisiones | Desarrollar e implantar un sistema informático | X | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | que soporte las decisiones estratégicas del Ministerio del Interior y Justicia y de la Dirección General de Custodia y Rehabilitación del Recluso | | | | |
| 10 | Nova Linux | Desarrollo de una nueva distribución de Linux para su utilización en la UCI. | | | | X |
| 10 | 23 y B | Realización de la intranet de la casona de 23 y B, una casa protocolo para los profesores de la Batalla de Ideas. | | X | | |
| 4 | Aduana | Desarrollo de varios subsistemas que se acoplaran al Sistema Único de Aduana | X | | | |
| 4 | Digitalización (Meta-datos) | Desarrollar una aplicación que permita digitalizar y procesar toda la documentación legal del Ministerio del Interior y Justicia de varios estado de la Republica Bolivariana de Venezuela. | X | | | |

Tabla 2: Aplicación de los métodos formales en los proyectos productivos.

## 2.6-. Conclusiones del capítulo.

1. Los estudiantes ya conocen muchos temas de los métodos formales, pero ni siquiera saben que estos forman la base para modelar o desarrollar los software si se quiere utilizar estas técnicas formales, y que se pueden vincular con los temas de la Ingeniería de Software para lograr una mayor calidad del producto.
2. Se demuestra que su inserción si bien es un poco trabajosa a los que están acostumbrados a los métodos convencionales, no es nada imposible de realizar y mucho más fácil si se trabaja por cada etapa de desarrollo.
3. Si bien es cierto que se dificulta en cierta medida el aprendizaje de estos métodos, no es de ningún modo imposible, además a pesar que todos los integrantes del equipo de desarrollo deben tener conocimientos sobre la materia, sólo menos de la mitad del equipo deben profundizar en estos contenidos.
4. En muchos de los proyectos productivos que se desarrollan en la universidad se pueden aplicar satisfactoriamente, ya sea en mayor o menor medida los conocimientos de los métodos formales.

**Conclusiones generales.**

1. Los conocimientos de los métodos formales en la Ingeniería de Software están muy poco difundidos en Cuba, que estas técnicas podrían aportar mucho al desarrollo de software ya que proporcionan una mayor seguridad y calidad en los sistemas y se eliminaría toda posibilidad de retraso en su desarrollo pero no están siendo explotadas.
2. Todas estas técnicas se ven frenadas por la creencia generalizada en la comunidad de desarrolladores que su aprendizaje e implementación son sumamente difíciles; y en verdad son un poco complejas pero no imposibles de de aprender y que la importancia de su aplicación viene dada por las grandes ventajas que nos ofrecen en la construcción de software.
3. Los estudiantes desconocen que en realidad, ya constan de conocimientos de métodos formales, pues muchos de estos temas se imparten en otras asignaturas de la carrera, por lo que no los utilizan.
4. Se pueden combinar satisfactoriamente las técnicas convencionales con los conocimientos formales para aplicarlos en la producción de software de la universidad.
5. El desconocimiento de estas técnicas imposibilita su aplicación en los proyectos productivos que se desarrollan en la universidad, pudiéndose utilizar, en mayor o menor medida, en el desarrollo de software, logrando así resultados mucho más satisfactorios.

## Recomendaciones

- Profundizar con el estudio de los métodos formales se realice un estudio más detallado en otras asignaturas de la carrera.
- Que se implementen estas técnicas formales para su utilización en la producción de la UCI.
- Que exista documentación suficiente y disponible en los diferentes repositorios con los que cuenta la universidad.
- Además, que los estudiantes y trabajadores dispongan de las herramientas para la aplicación de estas técnicas.

## Bibliografía

ANÓNIMO. *Algunos lenguajes de especificación*, 2007.

---. *El lenguaje CSP de Hoare*, 2003.

---. *Escuela de Ingenieria Informática de Sevilla*.

---. *Especificación de Arquitecturas Software*.

---. *Facultad de informática Universidad Politecnica de Madrid*.

---. *Formal Methods Europe*, 2004.

---. *Foundations of Computer Science university of Cambridge*, 4 septiembre 1998.

---. *MÉTODOS FORMALES EN LA INGENIERÍA DEL SOFTWARE.* .

---. *Proyecto FDISCS basado en métodos formales*, 7 marzo 2006.

---. *RAISE - Rigorous Approach to Industrial Software Engineering*, 22 Junio 2004.

---. *Rodin :Rigorous Open Development Environment for Complex Systems*.

---. *Semantics & Verification 2002*, 2002.

---. *Universidad de Murcia*.

BRENA, R. *AUTÓMATAS Y LENGUAJES*, 2003.

CANO, T. C. *Simulación de redes neuronales*.

CAÑADAS, J. J. *Introducción a los Métodos Formales en Ingeniería del Software*.

DARWIN SANTANA, P. R., EDUARDO FERNÁNDEZ NÚÑEZ. *TRABAJO DE INGENIERÍA DE SOFTWARE*, 2 noviembre 2006.

GOMÀRIZ, E. M. *Diseño de Sistemas Distribuidos*, 2007.

JACOB, I. *Métodos Formales en programación: ¿desmitificar para motivar?*

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

SOUSA, S. M. D. *Disciplina de Métodos Formais*, 2004.

## Referencias


1. SOLLA, A. G. *Diseño y verificación de sistemas distribuidos mediante la aplicación combinada de métodos formales.* 1999, marzo 2007, Disponible en: http://tvdi.det.uvigo.es/tesispdfs/tesis-alberto.pdf

2. PRESSMAN, R. S. *Ingeniería del Software: Un enfoque práctico.* 1985, vol. 2,

3. BOWEN, J. *Formal Methods.* junio 2006, Disponible en: http://www.comlab.ox.ac.uk/archive/formal-methods.html

4. ANÓNIMO. *¿Qué es la especificación formal?* diciembre 2006, Disponible en: http://www.cs.buap.mx/~jpalacio/gimf/esp-form.htm.

5. ---. *Departamento de Computación.* 2005, Disponible en: http://dc.exa.unrc.edu.ar/rio2007/anteriores/rio05.html

6. ---. *Lenguaje formal.* 12 marzo 2007, Disponible en: http://es.wikipedia.org/wiki/Lenguaje_formal

7. BOWEN, J. *The Z notation.* 15 September 2005, Disponible en: http://www.comlab.ox.ac.uk/archive/z.html.

8. LARSEN, P. G. *VDM Portal.* 2007, Disponible en: http://www.vdmportal.org/twiki/bin/view.

9. ANÓNIMO. *Lenguaje CSP* 2006, Abril 2007, Disponible en: http://webdia.cem.itesm.mx/ac/raulm/teaching/cc/notes/CSP.ppt

10. ---. *Larch family* Enero 2007, Disponible en: http://en.wikipedia.org/wiki/Larch_family

11. ---. *SLD Forum Society* Diciembre 2005, Disponible en: http://www.sdl-forum.org/SDL/index.htm

12. VALLECILLO, A. *Lección 2 Métodos Formales para Sistemas Abiertos.* abril 2007, Disponible en: http://www.lcc.uma.es/~av/Docencia/Doctorado/tema2.pdf

13. ANÓNIMO. *Red de Petri.* Abril 2007, Disponible en: http://es.wikipedia.org/wiki/Red_de_Petri

14. ---. *Real-Time Systems Group, The University of Pennsylvania.* 27 noviembre 1995, Disponible en: http://www.cis.upenn.edu/~lee/duncan/versa.html.

15. ---. *The DisCo Home Page* 2002, Disponible en: http://disco.cs.tut.fi/General.html

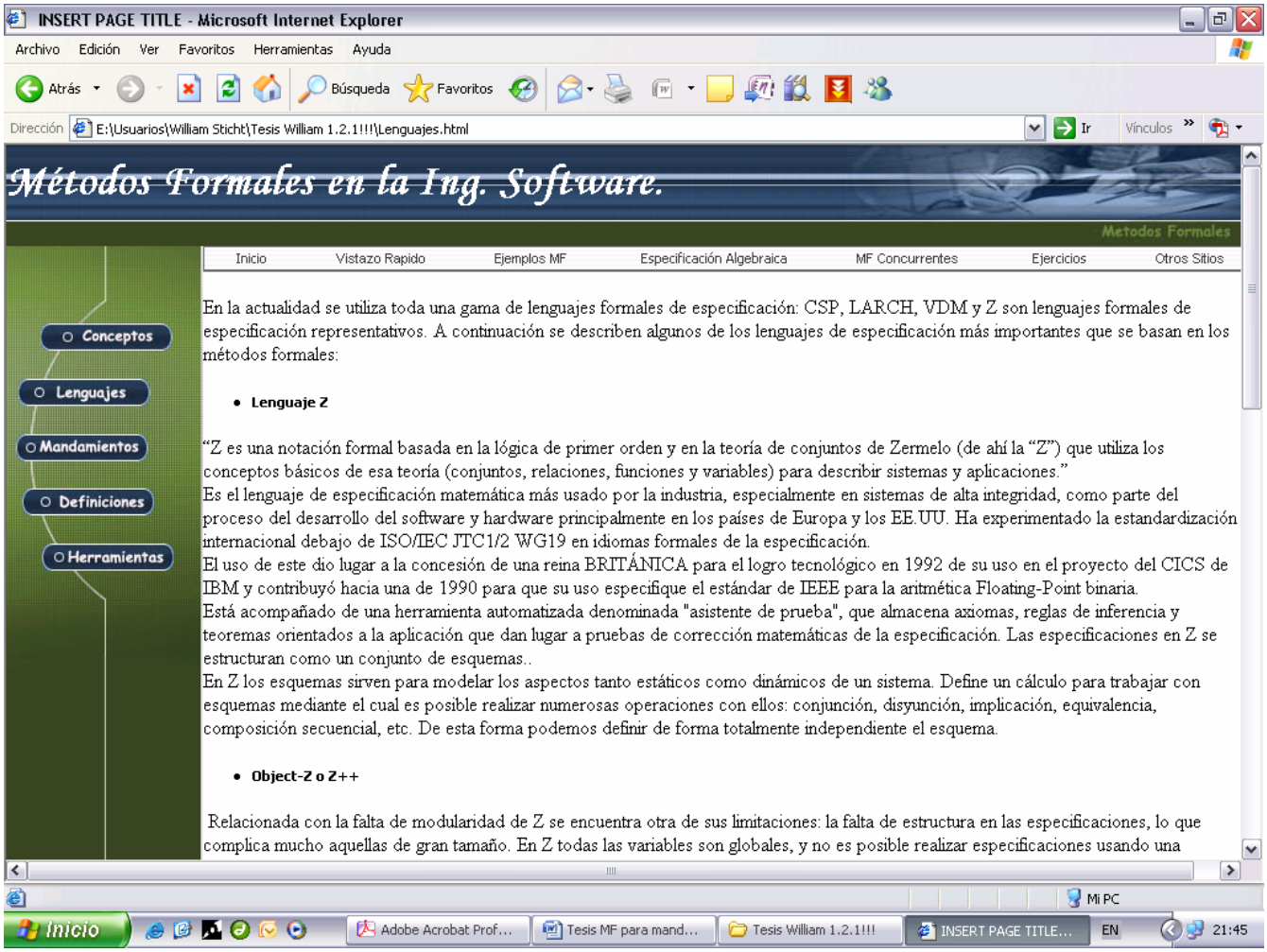
16. ---. *Cinderella SDL* 12 Abril 2006, Disponible en: http://www.cinderella.dk/index.htm.

## ANEXOS

Anexo 1: Imagen de la página principal del sitio.

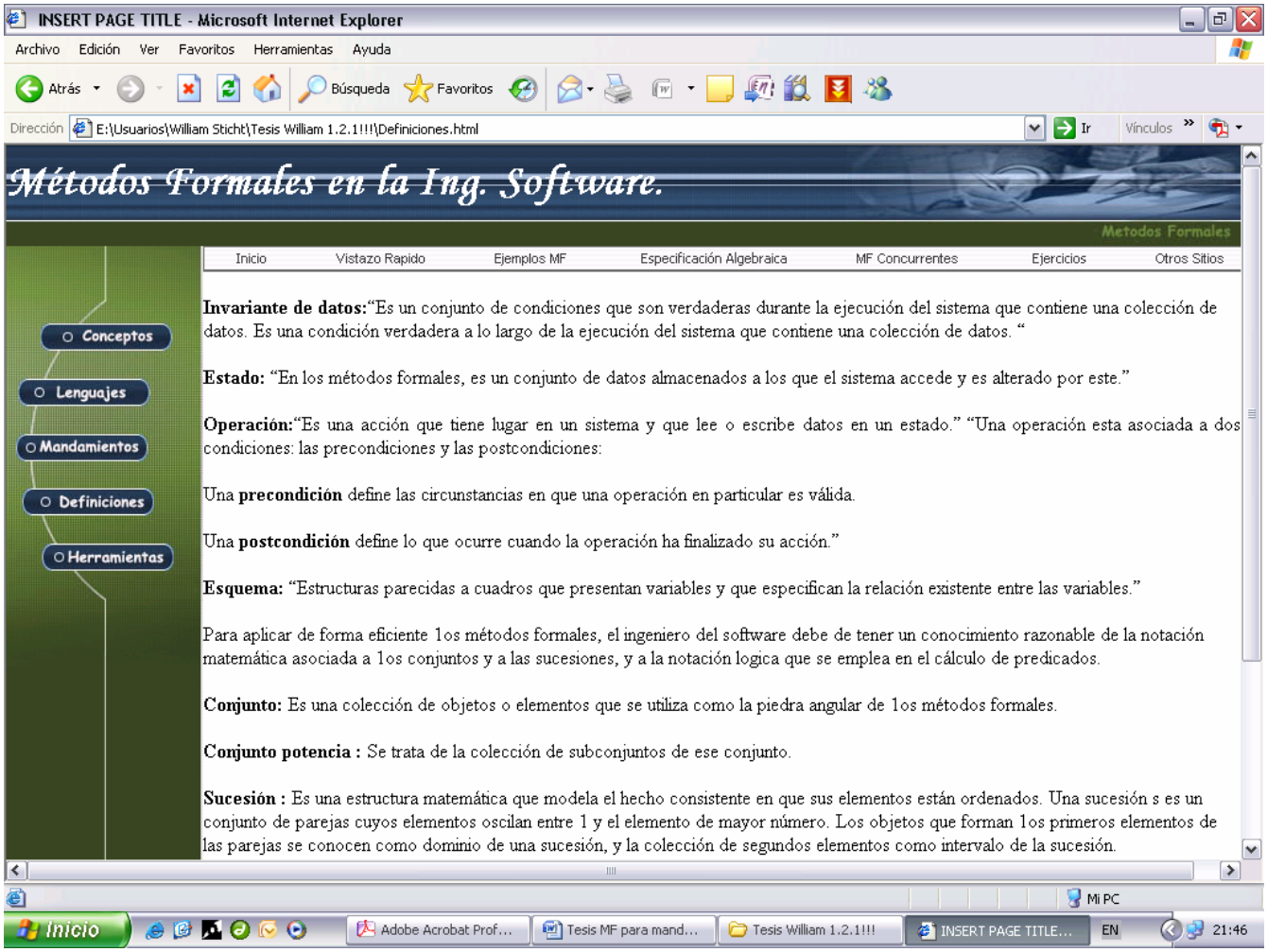
Anexo 2: Imagen de la página de concepto de los métodos formales.

## Anexo 3: Imagen de la página Lenguajes.

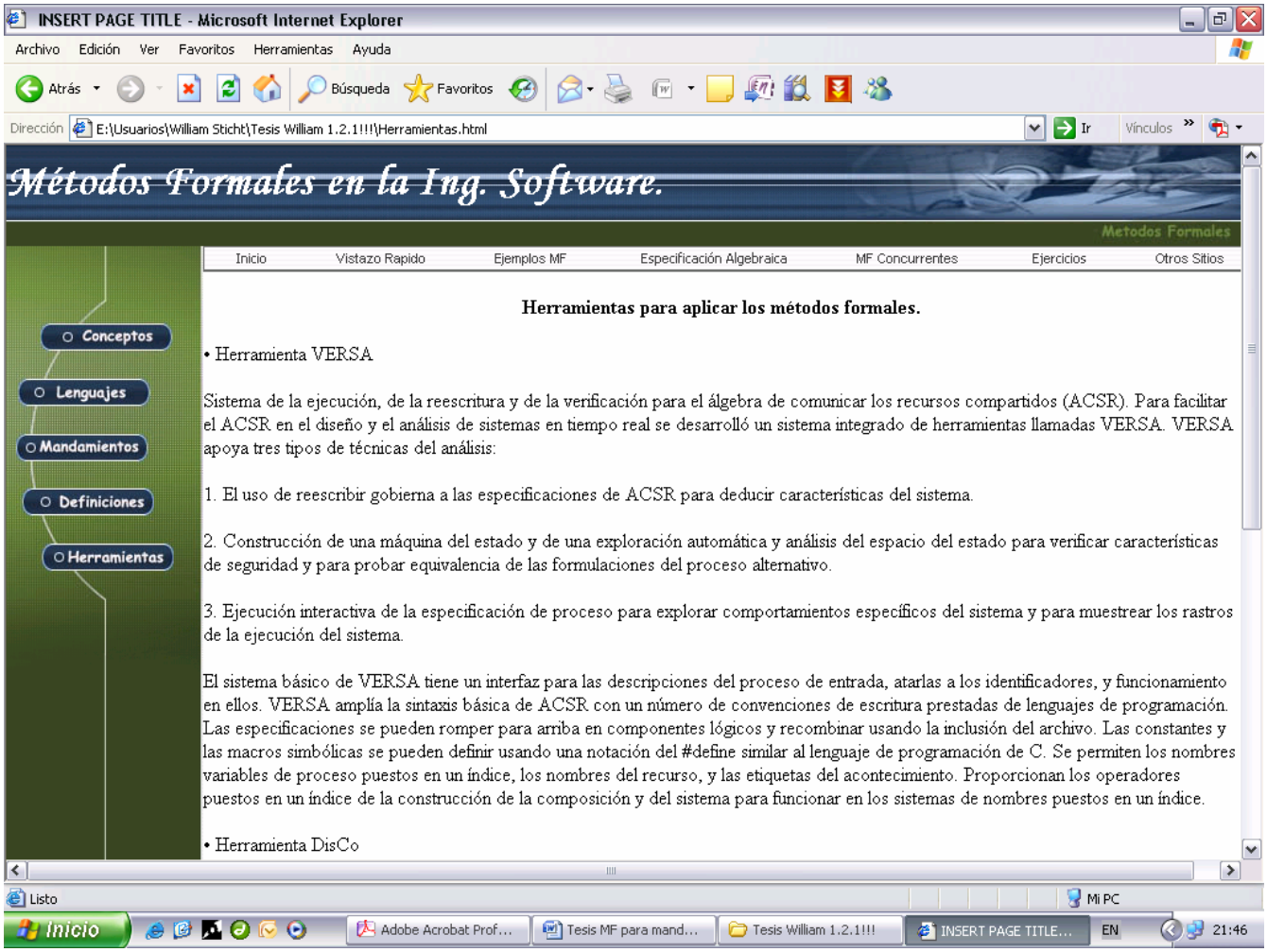


## Anexo 4: Imagen de la página Mandamientos.

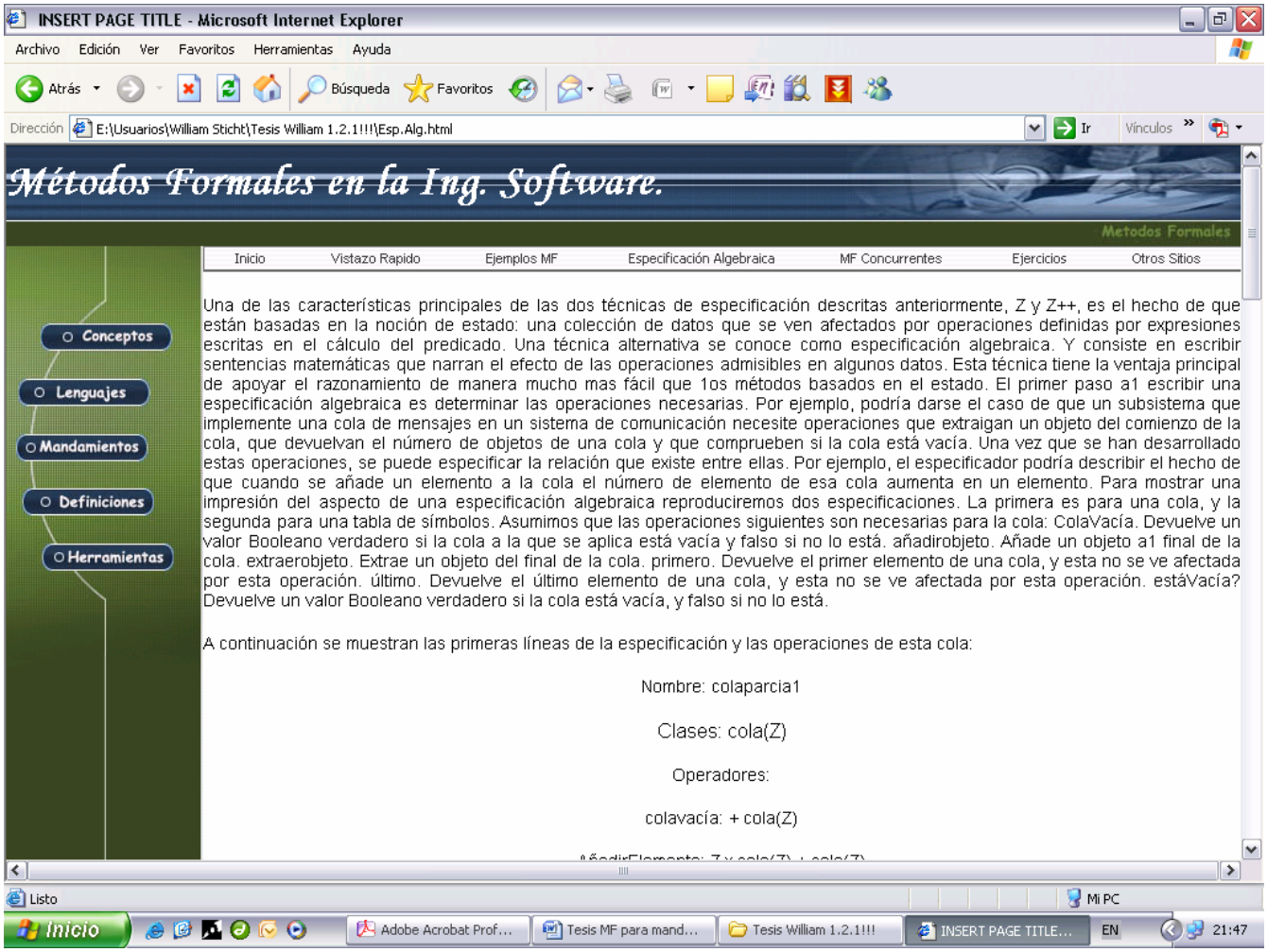
Anexo 5: Imagen de la página Definiciones.



## Anexo 6: Imagen de la página Herramientas.



## Anexo 7: Imagen de la página Enseñanza.



## GLOSARIO

1. Tecnología CASE: Define el uso de computadoras para el desarrollo de software en todo su ciclo de vida. Es un entorno de trabajo asistido por computadora que ayuda tanto a jefes de proyecto como a analistas y programadores en el diseño e implementación de todo tipo de software.
2. Ingeniería de Software: es una disciplina que se ocupa de la búsqueda de métodos, técnicas y herramientas para mejorar la calidad del software de un sistema informático.
3. Métodos formales: son técnicas de base matemática para describir las propiedades del sistema. Estos proporcionan marcos de referencia en el seno de los cuales las personas pueden especificar, desarrollar, verificar los sistemas de manera sistemática.
4. Invariante de datos: Es un conjunto de condiciones que son verdaderas durante la ejecución del sistema que contiene una colección de datos. Es una condición verdadera a lo largo de la ejecución del sistema que contiene una colección de datos.
5. Estado: En los métodos formales, es un conjunto de datos almacenados a los que el sistema accede y es alterado por este.
6. Operación: Es una acción que tiene lugar en un sistema y que lee o escribe datos en un estado.
7. Precondición: Define las circunstancias en que una operación en particular es válida.
8. Postcondición: define lo que ocurre cuando la operación ha finalizado su acción.
9. Esquema: Estructuras parecidas a cuadros que presentan variables y que especifican la relación existente entre las variables.
10. Lenguaje formal: es un conjunto de palabras (cadenas de caracteres) de longitud finita formadas a partir de un alfabeto (conjunto de caracteres) finito.