# Universidad de las Ciencias Informáticas

## “Facultad 1”

**Título: “Propuesta de técnicas de análisis de propiedades estructurales en la Universidad de las Ciencias Informáticas (UCI).”**

Trabajo de Diploma para optar por el titulo de Ingeniero en Ciencias Informáticas.


**Autora:**

Linet Castillo Clark

**Tutor:**

Ing. Dasiel Otero Dartayet

**Asesor:**

Lic. Ulises Llorente Pérez


“Año 50 de la Revolución”

Ciudad de La Habana

Junio 2008

*"Nadie puede quedar ajeno a los cambios que el mundo empieza a experimentar a partir de ahora."*

*Paulo Coelho*

## DECLARACION DE AUTORIA

Declaro ser autora de la presente tesis y reconozco 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 _____.

**Linet Castillo Clark**

_____

Firma del Autor

**Dasiel Otero Dartayet**

_____

Firma del Tutor

## OPINIÓN DEL TUTOR DEL TRABAJO DE DIPLOMA

Título**: "**Propuesta de técnicas de análisis de propiedades estructurales en la Universidad de las Ciencias Informáticas (UCI)".

Autor: Linet Castillo Clark

El tutor del presente Trabajo de Diploma considera que durante su ejecución el estudiante mostró las cualidades que a continuación se detallan.

Por todo lo anteriormente expresado considero que el estudiante está apto para ejercer como Ingeniero en Ciencias Informáticas; y propongo que se le otorgue al Trabajo de Diploma la calificación de ___.puntos.

**_____ días del mes de _____ del año 2008.**

_____

## DATOS DE CONTACTO

Tutor: Dasiel Otero Dartayet

Graduado de la UCI, año 2007. Instructor recién graduado.

Asesor: Ulises Lorente Pérez:

Licenciado en Educación. Especialidad Informática. Cursante de Maestría en Pedagogía Profesional en el Instituto superior Pedagógico “Héctor Alfredo Pineda Zaldívar” para la Educación Técnica y Profesional.15 años de experiencia como profesor, de ellos 8 en el IPVCE “Vladimir Ilich Lenin “Categoría docente de instructor desde 2006.Tutor, oponente y asesor de mas de una veintena de tesis de pregrado. Experiencia en impartición en varias ocasiones de las asignaturas:

- Metodología de la Investigación
- Metodología de la Enseñanza de la Informática
- Formación Pedagógica.
- Talleres de tesis
- Programación

Integrante de varios tribunales de tesis y corte de tesis.

## AGRADECIMIENTOS

*Agradezco:*

*A mis padres y mi hermano, mi más grande inspiración y los amores de mi vida, que nunca han dejado de creer en mi.*

*A mi tutor, por su ayuda y orientación.*

*Al profesor Ulises, por su orientación y sus consejos.*

*A la memoria de mis queridos abuelos, recuerdo imperecedero hasta el último día de mi vida.*

*A mi tía abuela Belkis, luchadora incansable por mi bienestar.*

*A mis tíos, Héctor Francisco, Ody, Clarita, Cari, Valia, Julia, mi tío Beto, por su comprensión y apoyo incondicional* .

*A Mati, por su amistad y ayuda en los momentos más difíciles.*

*A toda mi unida familia.*

*A mis amigas y compañeras: Yeni, Meylin, Edicta, Yanet, Adriana, Lyly, Yadira Corrales, Yadira Morales, Yaneyci, Arlenys y Maylen, gracias por su amistad y su ayuda en todo momento.*

*A mi antiguo compañero de tesis, Reynaldo.*

*A todas aquellas personas que siempre me ayudaron y alentaron en todas circunstancias.*

*A nuestro querido compañero y Comandante, Fidel, por su eterna confianza en nosotros los jóvenes.*

## DEDICATORIA

Dedico esta tesis a mis padres y mi hermano. A ellos gracias por ser mis guías en este largo camino, gracias por el apoyo, la comprensión y el amor que me profesan.

## RESUMEN

Cuando se comienza con el proceso y gestión de un software se pretende que este satisfaga debidamente las condiciones impuestas, y que cuente con la debida calidad y el funcionamiento necesario libre de errores y ambigüedades. En nuestros días se está llevando el proceso de ingeniería de software con todos los pasos de trabajo que este requiere para lograr el correcto funcionamiento de un sistema. No obstante, también existen técnicas que se están empleando recientemente, este es el caso de las técnicas de análisis de propiedades estructurales, que se basan en la creación de algoritmos matemáticos y la implementación de dichos algoritmos en el proceso de creación de un sistema de software. El presente trabajo tiene como objetivo la propuesta de estas técnicas de análisis de propiedades estructurales y su implementación en los proyectos productivos de la Universidad de las Ciencias Informáticas, en lo adelante UCI.

La investigación consta de tres capítulos:

El capítulo I: “Fundamentación teórica”, recoge los principales conceptos necesarios para un mejor entendimiento, conceptos como el de análisis, métodos formales y el de Arquitectura de Software, campo en el que se enmarca la investigación.

El capítulo II: “Técnicas de análisis de propiedades estructurales”, hace una caracterización de las técnicas que existen actualmente y a partir de un crítico análisis se elabora la propuesta de solución seleccionada por el autor.

**Palabras Claves:**

Arquitectura de Software, Métodos formales, Técnicas de análisis de propiedades estructurales.

# INDICE

# INTRODUCCION

Desde tiempos antiguos el hombre siempre soñó con tener una máquina que le propiciara hacer cálculos rápidos y exactos. Pero no fue hasta muchos años después que se hicieron descubrimientos que facilitaron tales acciones aunque aún existían sus dificultades a la hora de calcular números grandes, este fue el caso de la máquina de sumar de Pascal, inventada en 1642. Luego le sucedió Babbage, quien fue el primero en crear una computadora de uso general; la"Máquina de la diferencia". Pero no fue después hasta que en 1947, los científicos Bell, William Bardeen y Walter Brattain crearon el transistor, y a continuación Robert Noyce desarrolló el circuito integrado o chip. La evolución de la informática se ve presente en muchos aspectos de la vida. Actualmente la computadora es utilizada en casi todas las actividades que el hombre realiza.

El mundo actual impulsado por un gran avance científico en un marco socioeconómico neoliberal –globalizador y sustentado por el uso de las Tecnologías de la Información y la Comunicación (TIC) ha conllevado grandes cambios, manifestados de manera general en las actividades laborales, la educación, la cultura, la economía, la salud y en la defensa de un país. Estos cambios se han ido manifestando con el desarrollo de proyectos de software que han estado estrechamente unidos a estas esferas de la sociedad.

La informática estudia el tratamiento automático de la información utilizando dispositivos electrónicos y sistemas computacionales. (1)

La informática es la unión de diversas disciplinas: las ciencias de la computación (hardware), la programación y las metodologías para el desarrollo de software, la arquitectura de computadores, las redes de datos como Internet, la inteligencia artificial, así como determinados temas de electrónica.

El desarrollo de la informática en Cuba hizo que se trazaran planes de modernización de actividades que habían quedado notablemente retrasados,

como la bancaria y la hostelería. El sistema de Educación Superior organizó con éxito carreras para formar especialistas en informática y tecnologías afines, como la Licenciatura en Cibernética Matemática (Universidad de La Habana) y la Ingeniería en Sistemas Automatizados (ISPJAE). El resto de las especialidades y carreras que no la tenían, incorporan la informática como disciplina obligatoria. Otro ejemplo de aplicaciones científicas avanzadas es la bioinformática, la cual haya su expresión más elevada en el Centro de Bioinformática del Ministerio de Ciencia, Tecnología y Medioambiente (CITMA).

El Ministerio de la Informática y las Comunicaciones tiene como misión impulsar, facilitar y ordenar el uso masivo de servicios y productos de las tecnologías de la información, las comunicaciones, la electrónica y la automatización para satisfacer las expectativas de todas las esferas de la sociedad. Desarrolla actividades dirigidas a la comercialización de una amplia gama de productos, entre los que se destacan equipos electrodomésticos, equipos de telefonía, mobiliario, reciclaje, comercialización de cables y maquinarias para la industria, insumos para la gastronomía y el turismo, así como otros relacionados con la informática: equipos de computación, incluyendo sus partes, piezas y periféricos; energía y materiales eléctricos: baterías, protecciones eléctricas, componentes de sistemas solares; la electrónica; la automatización y sus accesorios y componentes.

También se concentran esfuerzos para desarrollar una Industria Nacional de Software, promoviendo la exportación de productos y servicios informáticos cubanos dirigidos principalmente a las aplicaciones multimedias, de alto valor agregado sobre Internet, fundamentalmente en las ramas de la medicina, la educación, la cultura y el deporte.

Todo especialista de ciencias computacionales ha practicado el desarrollo de software, lo cual se lleva a cabo con principios básicos de diseño como la abstracción, modularidad y encapsulamiento. Sin embargo, conforme la complejidad crece, estos principios son insuficientes para asegurar un producto exitoso que permita que los requerimientos y las expectativas de calidad sean alcanzados.

La Ingeniería de Software es una tecnología multicapa en la que se pueden identificar: los métodos (indican cómo construir técnicamente el software), el proceso ( fundamento de la Ingeniería de Software, es la unión que mantiene juntas las capas de la tecnología) y las herramientas (soporte automático o semiautomático para el proceso y los métodos). (2)

La Ingeniería de Software a gran escala es una actividad inherentemente compleja que involucra la creación y manipulación, por varias personas, de un gran número de artefactos frecuentemente intangibles y muy dinámicos. Éstos pueden incluir especificaciones de requerimientos, diseños de software de alto nivel, código fuente, información de pruebas, mantenimiento y necesidades de evolución posteriores a la implementación. Debido a la complejidad de la actividad, muchos de los esfuerzos se han dedicado a mejorar los pasos a seguir durante la construcción del software. Estándares ampliamente aceptados para alcanzar la calidad, como el conocido *Capability Maturity Model,* (CMM o Modelo de Capacidad de Madurez) se enfocan exclusivamente en el proceso. Aún a pesar de estas mejoras al proceso, no necesariamente se verá un cambio significativo correspondiente en el software que se construya. Un buen proceso no garantiza un buen producto. (5)

La Arquitectura de Software de un sistema de programa o de computación es la estructura del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente y las relaciones entre ellos. (2)

Todo sistema de software, desde el más pequeño hasta los sistemas multi-organizacionales tienen una arquitectura que puede variar en calidad. La arquitectura captura el conjunto de decisiones principales de diseño que se hacen sobre un sistema. Las decisiones de diseño son elecciones hechas pensando en cómo el sistema se desarrollará y cómo funcionará, y estas elecciones pueden incluir estructura, organización, funcionalidad, comportamiento, o más propiedades no funcionales como la usabilidad y estética de la interfaz de usuario. La importancia de estas decisiones de diseño varía para cada sistema de software y está en función de los interesados en el sistema, sus preocupaciones, y sus necesidades específicas. Una cuestión clave es que, a pesar de que la arquitectura es fundamentalmente una actividad centrada en el diseño, afecta a todo el ciclo de vida. Las decisiones de la arquitectura se

concentran en lo que es esencial en un sistema, la influencia de los requerimientos y las actividades importantes con el diseño colaborativo, el desarrollo e implementación del sistema, la planeación de su evolución y su adaptación.

Adquirir todos los beneficios de la Arquitectura de Software requiere una aplicación apropiada de la disciplina. Los sistemas con buena arquitectura son propensos al éxito, mientras sistemas con arquitecturas pobres tienen grandes probabilidades de fallo.

En Cuba, la política del Estado cubano ha sido la aplicación ordenada y masiva de la tecnología de la información y la comunicación en todas las esferas de la sociedad. Para garantizar una mayor eficiencia y eficacia en los procesos que contribuyan al bienestar del pueblo cubano.

En la UCI se llevan a cabo proyectos que garantizan además, la vinculación de Cuba con otros países, como es el caso de Venezuela, los cuales ingresan divisas al país. Proyectos en los que se encuentran vinculados estudiantes y profesores.

Para lograr que el cliente quede satisfecho, es necesario que el producto cumpla con los requerimientos y parámetros dados y que posea la eficacia deseada. Esta calidad tiene su mayor influencia en la Arquitectura de Software.

En la UCI se han presentado problemas a raíz del deficiente empleo de la Arquitectura de Software, lo que ha llevado a la siguiente **situación problémica**:

- No se llega a un buen refinamiento de requisitos funcionales lo que impide una mayor comprensión del problema para modelar la solución.
- No se verifica el cumplimiento de los requisitos tanto funcionales como no funcionales para el desarrollo óptimo del proceso.
- Incorrecto razonamiento sobre decisiones de diseño.
- El código a veces presenta errores que no es posible detectar fácilmente, lo que hace que se produzcan fallas en los sistemas.

Lo cual lleva al siguiente **problema científico**: ¿Cómo contribuir a mejorar la calidad de los proyectos productivos en la UCI?

El **objeto de estudio** es la Arquitectura de Software y el **campo de acción** donde se centra la investigación son las técnicas de análisis de propiedades estructurales.

Visto lo anterior se plantea la siguiente **hipótesis**:
Si se aplican las técnicas de análisis de propiedades estructurales apropiadas, los proyectos productivos en la UCI obtendrán una mayor calidad y eficiencia.

**Variable dependiente:**
Técnicas de análisis de propiedades estructurales

**Variable independiente:**
Calidad y eficiencia

El **objetivo general** que se persigue es: proponer técnicas de análisis de propiedades estructurales que contribuyan al desarrollo de los proyectos productivos en la UCI.

De lo cual se derivan los siguientes **objetivos específicos**:

- Analizar el estado del arte.
- Diagnosticar las técnicas estudiadas.
- Realizar una propuesta de técnicas de análisis.

El **posible resultado:** el desarrollo de las técnicas de análisis de propiedades estructurales en los proyectos productivos de la UCI.

Para darle cumplimiento a los objetivos mencionados se realizarán las siguientes **tareas**:

- Estudio de los Fundamentos formales de la Arquitectura de Software.
- Estudio de las técnicas de análisis de propiedades estructurales.
- Estudio de los métodos formales usados en la Arquitectura de Software.

- Propuesta de la mejor técnica de análisis de propiedades estructurales a desarrollar por los diferentes proyectos investigativos en la UCI.
- Diagnóstico del uso de las técnicas de análisis de propiedades estructurales
- Análisis del Estado del Arte.

Para lograr resultados satisfactorios se han aplicado **métodos científicos** como:

- El Método analítico sintético, ya que a partir de un estudio de teorías y documentos se pueden sintetizar los elementos más importantes y de mayor utilidad para el desarrollo del trabajo y en el momento final proponer una solución acertada.
- El Método histórico-lógico, con el que se ha estudiado la evolución y desarrollo de las técnicas de análisis de propiedades estructurales.
- La entrevista, que ha permitido establecer un diálogo con jefes y estudiantes vinculados a los proyectos productivos.

# CAPITULO I.FUNDAMENTACION TEORICA

## Introducción

La informática desempeña un papel fundamental en la Sociedad de la Información. Los millones de usuarios de esta nueva era informática que cada día entran en contacto, cooperan y comercian ,se tropiezan con las consecuencias derivadas de los fallos de software y las repercusiones que pueden tener estos fallos en las áreas donde la seguridad es crítica.

Por su naturaleza, las técnicas y herramientas automáticas deben descansar en fundamentos sólidos que permitan al usuario del sistema confiar en la aplicación bajo cualquier circunstancia.

En el mundo se han utilizado las técnicas de análisis de propiedades estructurales de muy pocas formas, una de ellas fue la que utilizaron Mara Alpuente y Salvador Lucas en su trabajo de la Introducción a la Ingeniería del Software Automática. (3)

Entre las técnicas de análisis que desarrollaron se encuentra la Técnica Estática y las Técnicas de Verificación Algorítmica y Declarativa y las propiedades fueron las de verificación formal, además dentro de la Inferencia de Tipos se encuentra la evaluación de prestaciones. También dentro de esta propiedad se encuentran algunas subpropiedades como son las de Aprendizaje Inductivo, Refinamiento Automático o Transformación de Programas, cada una de estas subpropiedades tiene una forma de realización que son:

- De propiedades a programas.
- De programas a propiedades.
- De datos a programas o propiedades.
- De propiedades a datos
- De propiedades a propiedades.

Pese a que los primeros trabajos sobre derivación formal de programas se remontan a los mismos orígenes de la programación como disciplina, estas técnicas han alcanzado una aceptación muy limitada, y los escasos ejemplos de aplicación práctica de este principio pasan por una automatización bastante parcial del proceso.

En los últimos años, se ha observado un creciente interés por los aspectos arquitectónicos del software. Estos aspectos se refieren a todo lo relativo a la estructura de alto nivel de los sistemas: su organización en subsistemas y la relación entre estos; la construcción de aplicaciones vista como una actividad fundamentalmente composicional, en la que se reutilizan elementos creados posiblemente por terceros; el desarrollo de familias de productos caracterizadas por presentar una arquitectura común y el mantenimiento y la evolución, entendidos como sustitución de unos componentes por otros dentro de un marco arquitectónico; etc.

## 1.1 Análisis

El análisis se refiere a la descomposición de un todo en sus distintos elementos constituyentes, con el fin de estudiar éstos de manera separada, para luego, en un proceso de síntesis, llegar a un conocimiento integral (4). El análisis se divide en: Análisis de Sistemas, Análisis Estructurado y Análisis Arquitectónico.

A pesar de que existan diversas maneras de realizar el análisis, todos tienen en común un conjunto de principios operativos:

- Debe representarse y entenderse el dominio de información de un problema.
- Deben definirse las funciones que debe realizar el software.
- Debe representarse el comportamiento del software (como consecuencia de acontecimientos extremos).
- Deben dividirse los modelos que representan información, función y comportamiento, de manera que se descubran los detalles por capas o jerárquicamente.
- El proceso de análisis debería ir desde la información esencial hasta el detalle de la implementación.

### 1.1.1 Análisis de Sistemas

El Análisis de Sistemas trata básicamente de determinar los objetivos y límites del sistema objeto de análisis, caracterizar su estructura y funcionamiento, marcar las directrices que permitan alcanzar los objetivos propuestos y evaluar sus consecuencias.

Dependiendo de los objetivos del análisis, se puede encontrar ante dos problemáticas distintas:

- Análisis de un sistema ya existente para comprender, mejorar, ajustar o predecir su comportamiento.
- Análisis como paso previo al diseño de un nuevo sistema-producto.

En cualquier caso, se pueden agrupar formalmente las tareas que constituyen el análisis en una serie de etapas que se suceden de forma iterativa, hasta validar el proceso completo:

- Conceptualización: Consiste en obtener una visión de muy alto nivel del sistema, identificando sus elementos básicos y las relaciones de éstos entre sí y con el entorno.
- Análisis funcional: Describe las acciones o transformaciones que tienen lugar en el sistema. Dichas acciones o transformaciones se especifican en forma de procesos que reciben entradas y producen salidas.
- Análisis de condiciones: Debe reflejar todas aquellas limitaciones impuestas al sistema que restringen el margen de las soluciones posibles. Estas se derivan a veces de los propios objetivos del sistema:
    - Operativas: como son las restricciones físicas, ambientales, de mantenimiento, de personal, de seguridad, etc.
    - De calidad: como fiabilidad, seguridad, convivencia, generalidad, etc.

  Sin embargo, en otras ocasiones las condiciones vienen impuestas por limitaciones en los diferentes recursos utilizables:

- Económicos: reflejados en un presupuesto.
- Temporales: que suponen plazos a cumplir.
- Humanos
- Metodológicos: que conllevan la utilización de técnicas determinadas.
- Materiales: herramientas disponibles, etc.

- **Construcción de modelos: Una de las formas más habituales y convenientes de** analizar un sistema consiste en construir un prototipo del mismo.
- Validación del análisis: A fin de comprobar que el análisis efectuado es correcto y evitar, en su caso, la posible propagación de errores a la fase de diseño, es imprescindible proceder a la validación del mismo. Para ello hay que comprobar los extremos siguientes:

  - El análisis debe ser consistente y completo.
  - Si el análisis se plantea como un paso previo para realizar un diseño, habrá que comprobar además, que los objetivos propuestos son correctos y realizables.

  Una ventaja fundamental que presenta la construcción de prototipos desde el punto de vista de la validación, radica en que estos modelos, una vez construidos, pueden ser evaluados directamente por los usuarios o expertos en el dominio del sistema para validar sobre ellos el análisis.

### 1.1.2 Análisis Estructural

El Análisis Estructural es la parte del proceso del proyecto que comprende el diseño, cálculo y comprobación de la estructura. Es esta una disciplina técnica y científica que permite establecer las condiciones de idoneidad de la estructura, respecto a su cometido o finalidad. Por tanto, tiene establecido su objeto en la estructura y su finalidad en el cálculo como comprobación de lo diseñado.

Muchos especialistas en sistemas de información reconocen la dificultad de comprender de manera completa sistemas grandes y complejos. El método de desarrollo del análisis estructurado tiene como finalidad superar esta dificultad por medio de:

- la división del sistema en componentes
- la construcción de un modelo del sistema

El método incorpora elementos tanto de análisis como de diseño. El análisis estructurado se concentra en especificar lo que se requiere que haga el sistema o la aplicación. Permite que las personas observen los elementos lógicos (lo que hará el sistema) separados de los componentes físicos (computadora, terminales, sistemas de almacenamiento, etc.). Después de esto se puede desarrollar un diseño físico eficiente para la situación donde será utilizado.

El análisis estructural tiene dos objetivos complementarios; en primer lugar lograr una representación lo mas exhaustiva posible del sistema estudiado, que permita, en una segunda fase, reducir la complejidad del sistema a sus variables esenciales.

Permite identificar los elementos de un problema y mostrar la manera en que estos guardan relación unos con otros. Parte del principio de que una variable no existe, sino, en virtud de las relaciones que guarda con las demás que conforman el sistema.

El análisis estructurado es un método para el análisis de sistemas manuales o automatizados, que conduce al desarrollo de especificaciones para sistemas nuevos o para efectuar modificaciones a los ya existentes. Éste análisis permite al analista conocer un sistema o proceso en una forma lógica y manejable, al mismo tiempo que proporciona la base para asegurar que no se omita ningún detalle pertinente.

### 1.1.3 Análisis arquitectónico

Existen muchos métodos de análisis arquitectónico que pueden ser aplicados en la arquitectura cubriendo los que se hacen de forma manual hasta los que son hechos totalmente automáticos. Por ejemplo, un método de análisis arquitectónico es una inspección en la que los interesados se reúnen y generalmente, ayudados por una lista de tareas u otra

agenda, revisan la arquitectura usando diferentes visualizaciones para ver el problema. En este sentido, el análisis puede ser más sobre el proceso que sobre las notaciones y las herramientas.

El objetivo final del análisis arquitectónico es el análisis totalmente automático: la arquitectura es definida, y una herramienta determina si la arquitectura alcanza cierta calidad o tiene cierta propiedad. Esto será posible en el grado de que dependa directamente de cómo la arquitectura será especificada. Muchas notaciones, especialmente las que se basan en modelos formales matemáticos, como el Wright[1], son desarrollados especialmente para facilitar ciertos análisis automatizados. Usualmente, estas notaciones están acompañadas de herramientas que apoyan este análisis. (5)

### 1.1.4 Análisis estático

El análisis estático es el estudio de las cosas que no están cambiando, sin embargo, en términos de software, esta definición es el estudio de el código fuente o binario, que no se está ejecutando. Se necesita un depurador para analizar el código fuente, pero se puede aprender mucho de código sin tener que ejecutar un programa. Dentro de los beneficios que reporta se encuentra que con sólo analizar todos los archivos de código fuente para un programa, se puede asegurar que el código fuente se adhiere a una norma de codificación predefinida. También se pueden detectar problemas comunes de funcionamiento, tales como llamar a un método varias veces a pesar de que el resultado que produce no cambia. Se puede incluso examinar las importaciones de cada clase para entender las clases dependientes. Nada de esto requiere que el programa se ejecute o incluso se compile. Al utilizar análisis estático se ahorra dinero y tiempo, y se obtiene una mayor calidad del código.

## 1.2 Arquitectura de Software

El término “arquitectura” aparece con mayor frecuencia en la literatura sobre Ingeniería de Software. El Proceso Unificado (Unified Process), presentado recientemente como el método

[1] La notación Wright permite realizar análisis estáticos acerca de la compatibilidad de las conexiones y la ausencia de bloqueos, pero no permite la creación dinámica de procesos, es decir, analiza la compatibilidad de las conexiones entre componentes y roles.

de desarrollo de software, asociado al Lenguaje Unificado de Modelado (UML), se define como centrado en la arquitectura.

Se puede considerar como arquitectura la estructura de alto nivel de un sistema de software, lo que incluye sus componentes, las propiedades observables de dichos componentes y las relaciones que se establecen entre ellos. Esta definición se centra en aspectos puramente descriptivos, y determina que cualquier sistema de software, o al menos, cualquiera que tenga una cierta complejidad, tiene una arquitectura, independientemente de si esta arquitectura está representada en algún lugar de forma explícita, o incluso, de si quienes desarrollaron el sistema son conscientes de ello.

Según el documento de IEEE Std 1471-2000, adoptada también por Microsoft: La Arquitectura de Software es la organización fundamental de un sistema, encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

En la notación UML se define la arquitectura como el conjunto de decisiones significativas acerca de la organización de un sistema de software; la selección de los elementos estructurales a partir de los cuales se compondrá el sistema y sus interfaces, junto con la descripción del comportamiento de dichas interfaces en las colaboraciones que se producen entre los elementos del sistema; la composición de esos elementos estructurales y de comportamiento para formar subsistemas de tamaño cada vez mayor; y el estilo o patrón arquitectónico que guía esta organización: los elementos y sus interfaces, las colaboraciones y su composición.

Aparejado a este concepto de arquitectura, surge la Arquitectura de Software, como disciplina, inscrita dentro de la Ingeniería de Software, que se ocupa del estudio de la arquitectura de los sistemas de software, y también como una de las tareas del proceso de desarrollo, enmarcada dentro de las actividades propias del diseño. (4) Los diseñadores han desarrollado un repertorio de métodos, técnicas, patrones y expresiones para estructurar

sistemas de software complejos. La Arquitectura de Software simplemente pretende dar rigor y hacer explícito todo este conocimiento.

En el siglo XXI, la Arquitectura de Software aparece dominada por estrategias orientadas a líneas de productos y por establecer modalidades de análisis, diseño, verificación, refinamiento, recuperación, diseño basado en escenarios, estudios de casos y hasta justificación económica, redefiniendo todas las metodologías ligadas al ciclo de vida en términos arquitectónicos. Todo lo que se ha hecho en ingeniería debe formularse de nuevo, integrando la Arquitectura de Software en el conjunto. La producción de estas nuevas metodologías ha sido masiva, y una vez más tiene como núcleo el trabajo del Instituto de Ingeniería de Software (Software Engineering Institute) en Carnegie Mellon. La aparición de las metodologías basadas en arquitectura, junto con la popularización de los métodos ágiles en general y la Programación Extrema[2] (Extreme Programming) en particular, han causado un reordenamiento del campo de los métodos, hasta entonces dominados por las estrategias de diseño “de peso pesado”.

Considerada como disciplina por mérito propio, la Arquitectura de Software es beneficiosa como marco de referencia para satisfacer requerimientos, una base esencial para la estimación de costos y administración del proceso y para el análisis de las dependencias y la consistencia del sistema.

La Arquitectura de Software ha resultado fundamental en un número respetable de escenarios, reduciendo costos, evitando errores, encontrando fallas, e implementando Sistemas de Misión Crítica[3]:

[2] Programación Extrema (XP) es una metodología de desarrollo ligera, que persigue el objetivo de aumentar la productividad a la hora de desarrollar programas.

[3] Los Sistemas de Misión Críticos son los que proporcionan la estructura vital de funcionamiento y las transacciones involucradas en realizar los movimientos de los números de cuentas. Estos sistemas pueden ser un grupo grande, una serie de servidores o una cúpula de almacenamiento de datos. Los sistemas de Misión Crítica deben convertirse y rodearse de una arquitectura de recuperación, estar protegidos contra riesgos y ser operados en un esquema al margen de los usuarios finales.

- La Arquitectura de Software representa un alto nivel de abstracción común que la mayoría de los participantes, si no todos, pueden usar como base para crear entendimiento mutuo, formar consenso y comunicarse entre sí. En sus mejores expresiones, la descripción arquitectónica expone las restricciones de alto nivel sobre el diseño del sistema, así como la justificación de decisiones arquitectónicas fundamentales.
- La Arquitectura de Software representa la encarnación de las decisiones de diseño más tempranas sobre un sistema, y esos vínculos tempranos tienen un peso fuera de toda proporción en su gravedad individual con respecto al desarrollo restante del sistema, su servicio en el despliegue y su vida de mantenimiento.
- Una descripción arquitectónica proporciona planos parciales para el desarrollo, indicando los componentes y las dependencias entre ellos. Por ejemplo, una vista en capas de una arquitectura documenta típicamente los límites de abstracción entre las partes, identificando las principales interfaces y estableciendo las formas en que unas partes pueden interactuar con otras.
- La Arquitectura de Software encarna un modelo relativamente pequeño, intelectualmente tratable, de la forma en que un sistema se estructura y sus componentes se entienden entre sí; este modelo es transferible a través de sistemas; en particular, se puede aplicar a otros sistemas que exhiben requerimientos parecidos y puede promover reutilización en gran escala. El diseño arquitectónico soporta reutilización de grandes componentes o incluso de frameworks en el que se pueden integrar componentes.
- La Arquitectura de Software puede exponer las dimensiones a lo largo de las cuales puede esperarse que evolucione un sistema. Quienes mantienen un sistema pueden comprender mejor las ramificaciones de los cambios y estimar con mayor precisión los costos de las modificaciones. Esas delimitaciones ayudan también a establecer mecanismos de conexión que permiten manejar requerimientos cambiantes de interoperabilidad, prototipo y reutilización.
- Las descripciones arquitectónicas aportan nuevas oportunidades para el análisis, incluyendo verificaciones de consistencia del sistema, conformidad con las restricciones

impuestas por un estilo, conformidad con atributos de calidad, análisis de dependencias y análisis específicos de dominio y negocios.

- La experiencia demuestra que los proyectos exitosos consideran una arquitectura viable como un logro clave del proceso de desarrollo industrial. La evaluación crítica de una arquitectura conduce típicamente a una comprensión más clara de los requerimientos, las estrategias de implementación y los riesgos potenciales.

### 1.2.1 Características de la Arquitectura de Software

La Arquitectura de Software se enmarca dentro de la Ingeniería de Software, y en particular del diseño de software, por lo que la primera de estas distinciones, se refiere a la arquitectura frente a los algoritmos y las estructuras de datos, y pretende fijar el papel de la Arquitectura de Software dentro del proceso de diseño. De este modo, el objetivo de la Arquitectura de Software no es el diseño de algoritmos o estructuras de datos, sino la organización a alto nivel de los sistemas de software, incluyendo aspectos como la descripción y análisis de propiedades relativas a su estructura y control global, los protocolos de comunicación y sincronización utilizados, la distribución física del sistema y sus componentes, etc. Junto a estos, también se presta especial atención a otros aspectos de carácter más general, relacionados con el desarrollo del sistema y su evolución y adaptación al cambio, como son los aspectos de composición, reconfiguración, reutilización, escalabilidad, mantenimiento, etc. Por tanto, el diseño arquitectónico se centra en lo que tradicionalmente se ha venido llamando diseño preliminar o diseño de alto nivel, frente al nivel de diseño detallado.

La segunda distinción se refiere a la naturaleza de los elementos objeto de estudio y, fundamentalmente, a las relaciones que se establecen entre ellos, y enfrenta las interacciones entre estos componentes con las relaciones de definición y uso que se utilizan para la definición de los módulos de código fuente del sistema. De esta forma, las relaciones de definición modularían el sistema en función de su código fuente, haciendo explícitas las relaciones de importación y exportación que indican dónde se define y dónde se usa ese código fuente. Sin embargo, desde el punto de vista de la Arquitectura de Software, el sistema

se divide en una serie de componentes, que no tienen por qué coincidir con los módulos de compilación. Estos componentes realizan conmutaciones y almacenamiento de datos, y llevan a cabo diversas interacciones unos con otros durante la ejecución del sistema.

La tercera distinción se establece entre los métodos arquitectónicos y los métodos de desarrollo de software, como los orientados a objetos o los estructurados. Mientras el objetivo de estos últimos es proporcionar un camino entre el espacio del problema y el de la solución, la Arquitectura de Software se centra únicamente en el que se podría denominar como espacio de los diseños arquitectónicos, preocupándose de cómo guiar las decisiones a tomar en este espacio, decisiones que se basan en las propiedades de los diferentes diseños arquitectónicos y su capacidad para resolver determinados problemas. No obstante, ambos métodos están estrechamente relacionados y se complementan: detrás de la mayoría de los métodos de desarrollo hay un estilo arquitectónico preferido, mientras que el desarrollo y uso de nuevos estilos arquitectónicos lleva normalmente a la creación de nuevos métodos de desarrollo que exploten sus características.

La última distinción se establece dentro del propio campo de la arquitectura, y enfrenta a los estilos arquitectónicos con las instancias de los mismos. Una instancia arquitectónica se refiere a la arquitectura de un sistema concreto, mientras que un estilo arquitectónico define las reglas generales de organización y las restricciones en la forma y la estructura de un grupo numeroso y variado de sistemas de software.

### 1.2.2 Conceptos fundamentales de Arquitectura de Software:

- Estilos:

Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales. Los estilos enlazan componentes, conectores, configuraciones y restricciones. La descripción de un estilo se puede formular en lenguaje natural o en

diagramas, pero lo mejor es hacerlo en un lenguaje de descripción arquitectónica o en lenguajes formales de especificación. A diferencia de los patrones de diseños, que son centenares, los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares, como máximo. Las arquitecturas complejas o compuestas resultan del agregado o la composición de estilos más básicos. Algunos estilos típicos son las arquitecturas basadas en flujo de datos, las peer-to-peer[4], las de invocación implícita, las jerárquicas, las centradas en datos o las de intérprete-máquina virtual. (6)

- Lenguajes de descripción arquitectónica (ADLs):

Son un conjunto de propuestas de variado nivel de rigurosidad, casi todas ellas de extracción académica, que fueron surgiendo desde comienzos de la década de 1990 hasta la actualidad, más o menos en contemporaneidad con el proyecto de unificación de los lenguajes de modelado. Uno de estos lenguajes es el UML.

UML o Lenguaje Unificado de Modelado es una notación para especificar, visualizar y documentar sistemas de software desde la perspectiva de la orientación a objetos. UML postula un proceso de desarrollo iterativo, incremental, guiado por casos de uso y centrado en la arquitectura, maneja conceptos utilizados también por la Arquitectura de Software, como son los de interfaz, componente o conexión, y que los mecanismos de extensión disponibles pueden utilizarse para definir otros conceptos no contemplados o para establecer restricciones que definan de forma más precisa la semántica de estos conceptos. (7).

Es necesario especificar antes de seguir, los distintos conceptos:

- Componente: representan unidades de computación o de almacenamiento de datos.
- Conectores: Son utilizados para modelar las interacciones entre componentes y las reglas que gobiernan dichas interacciones.

El objetivo de los lenguajes es proporcionar un vehículo de expresión para las nociones intuitivas y prácticas de sus usuarios, en este caso los arquitectos de software. Los ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las

[4] La red peer-to-peer o de punto a punto hace referencia a una red que no tiene clientes ni servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y servidores de los demás nodos de la red.

aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento.

- Frameworks y Vistas:

Un frameworks representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio.

Una vista es, para definirla sintéticamente, un subconjunto resultante de practicar una selección o abstracción sobre una realidad, desde un punto de vista determinado. A pesar de que la mayoría de los frameworks reconocen entre tres y seis vistas, muchos arquitectos de la corriente principal evitan hablar de vistas, porque cuando proliferan se hace necesario, o bien elaborar lenguajes formales específicos para tratar cada una de ellas, o bien multiplicar salvajemente las extensiones del lenguaje unificado. Sin duda las vistas son una simplificación conveniente, o más bien un principio de orden; pero su abundancia y sus complicadas relaciones recíprocas generan también nuevos órdenes de complejidad. Sin embargo, los arquitectos practicantes las usan de todas maneras, porque simplifican la visualización de sistemas complejos.

En 1995 Philippe Kruchten propuso su célebre modelo “4+1”, vinculado al Rational Unified Process (RUP):

- La vista lógica, que comprende las abstracciones fundamentales del sistema a partir del dominio de problemas.
- La vista de procesos: el conjunto de procesos de ejecución independiente a partir de las abstracciones anteriores.
- La vista de despliegue: un mapeado del software sobre el hardware.
- La vista de implementación: la organización estática de módulos en el entorno de desarrollo.

- La vista de casos de uso: considera todos los anteriores en el contexto de casos de uso. Lo que académicamente se define como Arquitectura de Software concierne a las dos primeras vistas.

El modelo 4+1 vistas arquitectónicas se percibe hoy como un intento de reformular una arquitectura estructural y descriptiva en términos de objetos y de UML.

- Procesos y Metodologías:

Llegando al territorio de la metodología, hay que decir que durante varios años la Arquitectura de Software discurrió sin elaborarlas más que circunstancialmente, como si se estimara compatible con las prácticas establecidas en Ingeniería de Software, cualesquiera fuesen: RUP, RAD, RDS, ARIS, PERA, CIMOSA, GRAI, GERAM, CMM. Hoy en día la metodología dominante en la industria es tal vez el Modelo de Madurez de la Capacidad (CMM), aunque el Instituto de Ingeniería de Software no la considera formalmente como tal.

Desde 1998 y cada vez con mayor intensidad, sin embargo, el Instituto de Ingeniería de Software y otros organismos comenzaron a elaborar métodos específicos de procesos de ingeniería que sistematizan el rol de la arquitectura en la totalidad del proceso, desde la creación de los requerimientos hasta su terminación. Algunos de esos métodos son: Arquitectura Basada en Diseño (ABD), Método de Análisis de la Arquitectura de Software (SAAM), Talleres de Atributo de la Calidad (QAW), Método de Diseño del Atributo de la calidad orientado a la Arquitectura de Software (QASAR), Método de Análisis de Costo-Beneficio (CBAM), Método de Análisis de la Familia de la Arquitectura (FAAM), y Método de Análisis de Comparación de la Arquitectura de Software (SACAM). (6)

En lo que respecta a la estrategia arquitectónica de Microsoft, el marco general de Microsoft Solutions Framework, versión 3, no se inclina sobre metodologías específicas y da cabida a una gran cantidad de variantes. Abundan en MSF 3, referencias a métodos rápidos y ágiles.

- Abstracción

La abstracción consiste en extraer las propiedades esenciales, identificar los aspectos importantes, o examinar selectivamente ciertos aspectos de un problema, posponiendo o ignorando los detalles menos sustanciales e irrelevantes.

La idea de abstracción forma parte de lo que acaso sea la pieza conceptual más importante de la Arquitectura de Software, el concepto de estilo; un estilo se identifica a grandes rasgos o, como se dice habitualmente, en un estilo "menos es más". Las ventajas incluyen mejor reutilización, mejores análisis, menor tiempo de selección y mayor interoperabilidad.

- Escenarios

Los escenarios se dividen en dos categorías: la primera categoría la constituyen los casos de uso (secuencias de responsabilidades) y la segunda los casos de cambio (modificaciones propuestas al sistema). Los escenarios, han sido básicamente técnicas que se implementan en la creación de los requerimientos, particularmente en relación a los operadores de sistemas. Típicamente, la técnica comienza instrumentando sesiones de brainstorming (lluvia de ideas). También se han utilizado escenarios como método para comparar alternativas de diseño. Los escenarios describen una utilización anticipada o deseada del sistema. Se los considera útiles para analizar una vista determinada o para mostrar la forma en que los elementos de múltiples vistas se relacionan entre sí. Pueden concebirse también como una abstracción de los requerimientos más importantes de un sistema. Los escenarios se describen mediante texto común en prosa utilizando lo que se llama un script[5] y a veces se describen mediante dibujos, como por ejemplo los diagramas de interacción de objetos. Se acostumbra utilizar UML (en el contexto de 4+1 vista) no tanto como recurso de modelado que después generará alguna clase de código, sino como instrumento de dibujo más o menos informal; pero los propios manuales de UML y los expertos mundiales en casos de uso (David

[5] Un script es un guión o un conjunto de instrucciones, que permite la automatización de tareas creando pequeñas utilidades. Usualmente son archivos de texto. También se considera como un script una alteración o acción a una determinada plataforma.

Anderson, Martin Fowler, Alistair Cockburn) recomiendan desarrollar los escenarios de requerimiento en texto, no en diagramas.

- Patrones

Un patrón es una solución probada que se puede aplicar con éxito a un determinado tipo de problemas que aparecen repetidamente en algún campo. Los patrones no se proponen descubrir ni expresar nuevos principios de la Ingeniería de Software. Todo lo contrario: intentan codificar el conocimiento, las expresiones y los principios ya existentes: cuanto más trillados y generalizados, tanto mejor. (8) El establecimiento de estos patrones comunes es lo que posibilita el aprovechamiento de la experiencia acumulada en el diseño de aplicaciones. Un diseñador experimentado producirá diseños más simples, robustos y generales, y más fácilmente adaptables al cambio. Por otra parte, un buen diseño no debe ser específico de una aplicación concreta, sino que debe basarse en soluciones que han funcionado bien en otras ocasiones.

En el diseño de alto nivel se encuentran los patrones arquitectónicos. Estos patrones son esquemas de organización de los sistemas de software que determinan cuál va a ser la estructura de los sistemas, mediante el establecimiento de su división en subsistemas, indicando las responsabilidades de cada uno de estos subsistemas y las reglas y criterios que rigen las relaciones entre ellos. Ejemplos de patrones arquitectónicos lo constituyen los patrones Modelo-Vista-Controlador, Tuberías y Filtros y Modelo de 3 Capas.

Por otra parte en el diseño de bajo nivel, o diseño detallado se enmarcan los patrones de diseño, estos, a diferencia de los patrones arquitectónicos no afectan la estructura general de una aplicación, sino sólo a una parte puntual de la misma. Ofrecen todo un catálogo de patrones básicos aplicables a la programación orientada a objetos en general. Ejemplo de patrones de diseño lo constituyen los patrones GRASP (Patrones Generales de Software para asignar responsabilidades).

Entre las ventajas de los patrones de diseño se encuentran que son soluciones simples y técnicas que se aplican a problemas muy comunes que aparecen en el diseño orientado a objetos, facilitando la reutilización del diseño. También, es necesario referirse a su carácter fundamentalmente práctico, que indica como resolver el problema desde el punto de vista técnico de la orientación a objetos. Entre sus inconvenientes se indicará, en primer lugar, que son soluciones concretas a problemas concretos. Un catálogo de patrones es un libro de recetas de diseño, y aunque existen clasificaciones de patrones, todos ellos son independientes, lo que hace difícil, dado un problema determinado, averiguar si existe un patrón que lo resuelve. En segundo lugar, el uso de un patrón no se refleja claramente en el código de la aplicación. A partir de la implementación de un sistema es difícil determinar qué patrones de diseño se han utilizado en la misma. Por último, si bien permiten reutilizar el diseño, es mucho más difícil reutilizar la implementación del patrón. Este describe clases que juegan papeles genéricos, explicadas normalmente con ayuda de un ejemplo, pero su implementación, adaptándolo a las peculiaridades de una aplicación en particular, consistirá en clases concretas, dependientes del dominio de aplicación. No existe un lenguaje de especificación de patrones que permita su reutilización por medio de la instanciación de una descripción genérica del mismo.

### 1.2.3 Especificación de la Arquitectura de Software.

Uno de los aspectos fundamentales en el diseño de cualquier sistema de software de cierta complejidad es su estructura o arquitectura, representada por un conjunto de elementos computacionales o componentes y una serie de conexiones o interacciones entre estos componentes.

La aplicación de técnicas y métodos específicos durante esta fase constituye un campo al que se le dedica un interés cada vez mayor. No sólo se ha venido estableciendo una colección de métodos, técnicas, esquemas y expresiones para describir la arquitectura de un sistema de software, sino que se tiende al desarrollo de entornos que permitan la reutilización efectiva de los componentes de un sistema, y de la propia arquitectura o configuración del mismo. La idea básica es desarrollar patrones o esquemas de arquitecturas, de forma que el desarrollo de

nuevos sistemas de software se vea facilitado mediante la instanciación de alguno de estos patrones comunes.

La aplicación de métodos y lenguajes formales se ha mostrado especialmente útil en las etapas iniciales del proceso de desarrollo, tanto para la especificación como para la validación de propiedades.

Aunque formalismos como el cálculo pi, enfocado en la descripción y análisis de sistemas concurrentes (estos sistemas se representan mediante colecciones de procesos que interactúan por medio de enlaces), cuya topología es dinámica, apenas se usan en el desarrollo de sistemas de software industriales, el desarrollo de lenguajes de alto nivel y herramientas que les sirvan de soporte puede servir para acortar la distancia entre los fundamentos formales y su aplicación práctica. Este es el punto de vista que se ha adoptado en el desarrollo de LEDA: Lenguaje de Especificación para la Descripción y Validación de la Arquitectura, que utiliza el cálculo pi como base formal. Esto permite el desarrollo y aplicación de técnicas de análisis específicas para comprobar la consistencia de un diseño o demostrar propiedades relativas a su operatividad. Por ejemplo, es posible comprobar la compatibilidad de una arquitectura, es decir, si sus componentes presentan comportamientos compatibles y pueden ser combinados de forma segura para formar el sistema. Por otro lado, la reutilización del software se fomenta si se es capaz de comprobar si un cierto componente puede ser utilizado en un nuevo sistema donde se requiere un comportamiento similar. De nuevo surge la noción de compatibilidad entre componentes. La compatibilidad no exige que el comportamiento de los componentes involucrados coincida exactamente, puesto que normalmente se quiere conectar componentes que encajan sólo parcialmente, incrementando las posibilidades de reutilización.

Finalmente, la existencia de una noción de extensión o herencia de componentes en LEDA, permite considerar las especificaciones como patrones genéricos de arquitecturas o frameworks, que pueden ser extendidas y reutilizadas, adaptándolas a nuevos requisitos.

En la Arquitectura de Software muchos lenguajes de descripción de la arquitectura presentan una base formal, lo que permite el análisis y verificación de los sistemas descritos.

## 1.3 Métodos Formales

Un método formal es cualquier actividad relacionada con representaciones matemáticas del software.

Los métodos formales representan un conjunto de tendencias de desarrollo de software y hardware en donde la especificación, verificación y diseño de componentes se realiza mediante notaciones, lenguajes, herramientas y técnicas basadas en teorías con sólida fundamentación matemática.

Las notaciones y lenguajes formales permiten plantear claramente los requisitos del sistema y generan especificaciones que nos muestran el “qué debe hacer el sistema” y no el “cómo lo hace “.Las especificaciones implementan el sistema casi de manera automática.

Estos métodos 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. La teoría de conjuntos y las notaciones lógicas se utilizan para crear una sentencia clara de hechos (o de requisitos). Esta especificación matemática se puede analizar para comprobar si es correcta o constante.

Los ingenieros son los especializados para crear una especificación formal. El primer paso para la aplicación de los métodos formales es la definición del invariante de datos, el estado y las operaciones para el funcionamiento de un sistema. El invariante de datos es un conjunto de condiciones verdaderas a lo largo de la ejecución del problema que contiene una colección de datos. Define lo que está garantizado que no cambiará.

Los datos almacenados forman el estado[6]; y las operaciones, son las acciones que tienen lugar en un sistema, a medida que se leen o se escriben datos en un estado. Una operación se asocia a dos condiciones: una precondición y una poscondición. La notación y la heurística asociadas con los conjuntos y especificaciones constructivas (operadores de conjuntos, operadores lógicos y sucesiones) forman la base de los métodos formales.

Los métodos formales difieren en la manera y tiempos de cada una de las fases del ciclo de vida del software. Requieren mayor tiempo en el desarrollo de la especificación y la construcción de diseños correctos, lo que aumenta el tiempo de las fases de análisis y diseño; mientras que los métodos usados, correctos por construcción, disminuyen el tiempo de verificación del software, al requerir una cantidad de casos de prueba mucho más reducido que cubre todo el panorama de prueba, a diferencia de validaciones en base a simulaciones que son incompletas e ineficientes.

Los métodos formales requieren de un nivel avanzado en las matemáticas, por lo que se hace necesario que cada miembro del equipo de desarrollo sea capaz de definir y entender especificaciones por sí mismos.

Los métodos formales pueden ser reconocidos como una nueva metodología, y dentro de las razones se pueden destacar las siguientes: la generación de sistemas correctos en ambientes en donde un fallo puede resultar crítico para la estabilidad de un sistema, los beneficios de un costo ostensiblemente menor, el aumento en la documentación, y la reducción de errores en los procesos de desarrollo de software.

Actualmente, los métodos formales han sido principalmente utilizados en las etapas de análisis y diseño. Existen algunos casos documentados en donde el proceso completo de desarrollo ha sido guiado desde una perspectiva formal: el sistema para tráfico aéreo CDIS, el sistema aeronáutico para Lookheed y la verificación de procesadores.

[6] Es 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.

Los métodos formales se clasifican en base al modelado matemático en:

- Especificaciones basadas en lógica de primer orden y teoría de conjuntos: permiten especificar el sistema mediante un concepto formal de estados y operaciones sobre estados. Los datos y relaciones se describen en detalle y sus propiedades se expresan en lógica de primer orden. La semántica de los lenguajes está basada en la teoría de conjuntos. Los métodos de este tipo más conocidos son: Z, VDM y B.
- Especificaciones algebraicas: proponen una descripción de estructuras de datos estableciendo tipos y operaciones sobre esos tipos.
- Especificación de comportamiento:
    - Métodos basados en álgebra de procesos: modelan la interacción entre procesos concurrentes. Esto ha potenciado su difusión en la especificación de sistemas de comunicación (protocolos y servicios de telecomunicaciones) y de sistemas distribuidos y concurrentes. Los más conocidos son: CCS, CSP y LOTOS.
    - Métodos basados en Redes de Petri: una Red de Petri es un formalismo basado en autómatas, es decir, un modelo formal basado en flujo de información. Permiten expresar eventos concurrentes. Los formalismos basados en Redes de Petri establecen la noción de estado de un sistema mediante lugares que pueden contener marcas. Un conjunto de transiciones (con precondiciones y poscondiciones) describe la evolución del sistema, entendido como la producción y consumo de marcas en varios puntos de la red.
    - Métodos basados en lógica temporal: se usan para especificar sistemas concurrentes y reactivos. Los sistemas reactivos son aquellos que mantienen una continua interacción con su entorno respondiendo a los estímulos externos y produciendo salidas en respuesta a los mismos, por lo tanto el orden de los eventos en el sistema no es predecible y su ejecución no tiene por qué terminar.

## 1.4 Verificación formal

La Verificación formal estudia los fundamentos teóricos y la implementación de técnicas de verificación de sistemas computacionales La verificación formal consta de tres etapas:

- Modelación: Donde se construye un modelo matemático de los posibles comportamientos del sistema
- Especificación: Se especifica en un lenguaje formal los comportamientos deseables del sistema.
- Verificación: Se comprueba que el modelo satisfaga la especificación.

Es formal porque el modelo y la especificación son objetos matemáticos y es verificación porque el análisis responde con certeza si la especificación se cumple o no (otros tipos de verificación incluyen testeo y simulación, que no son exhaustivos).

La verificación puede ser: automatizada e interactiva. La verificación automatizada se realiza de forma algorítmica. La idea es simular exhaustivamente todos los posibles comportamientos del sistema en busca de fallas. A este tipo de verificación también se le llama Model Checking.

El Model Checking es un método general con aplicaciones en verificación de hardware, verificación de sistemas de software, verificación de protocolos criptográficos, etc. Responde con certeza si la especificación es cierta o no en el sistema; esta información resulta muy beneficiosa para solucionar los problemas del sistema. Dentro de las limitaciones del Model Checking se encuentra que trabaja sobre un modelo del sistema. Su fuerza radica en detectar errores, no chequear corrección. Más adelante en el próximo capítulo se especifica sobre esta técnica.

La Verificación interactiva, por otra parte, consiste en probar un teorema desde un conjunto de axiomas. Al contrario de la verificación automatizada, sólo puede ser realizada por expertos en lógica.

- Especificación formal**:**

Una 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.

El desarrollo de métodos formales se concreta en la creación de herramientas, las cuales se pueden caracterizar rigurosamente de la siguiente manera:

Herramienta = Lenguaje Formal + Inferencia mecánica

El lenguaje formal se fundamenta ya sea en alguna lógica, cálculo o en álgebra, con una sintaxis y semántica determinadas. El lenguaje permite expresar propiedades de un dominio matemático de manera clara y no ambigua. La amplitud del dominio matemático establece la capacidad expresiva del lenguaje. Un lenguaje formal puede estar completamente basado en la lógica, álgebra ó cálculo que lo sustenta, o tratarse de un fragmento sintáctico, dentro del cual se puedan expresar propiedades útiles. El lenguaje formal está constituido por:

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

La inferencia mecánica se refiere al sustento operacional, generalmente en términos de la teoría de pruebas para una lógica u operaciones de reducción en un álgebra. Se pretende que las propiedades fundamentales de un sistema que están siendo modeladas por el lenguaje formal puedan ser demostradas: la construcción de pruebas formales constituye el vehículo principal para reflejar las propiedades del sistema que está siendo modelado, identificando posibles errores de diseño o inconsistencias; más aún, la construcción de pruebas se vuelve trascendente cuando permite definir paradigmas abstractos de cómputo: cómputo como reducción de pruebas y cómputo como búsqueda de pruebas. En particular, para cuando el lenguaje formal está fundamentado en una lógica, su inferencia mecánica se basa en un demostrador automático de teoremas que puede o no estar apoyado por un constructor de modelos.

El uso de métodos formales se toma como una fase separada entre la formulación de 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, o en otras palabras, producir especificaciones eficientemente ejecutables. Es de especial mención los avances en cuanto a la generación semántica de compiladores y máquinas abstractas.

Para saber si una especificación es correcta se aplican pruebas lógicas a cada función del sistema, debido a que los métodos formales utilizan la matemática discreta como mecanismos de especificación.

- Modelación**:**

La modelación describe la interacción y el control del sistema. No intenta describir manipulación de datos ni estructuras de datos complejas. Un sistema puede ser modelado a diferentes niveles de abstracción y por tanto, tener diferentes modelos. Dentro de las formas de describir modelos se incluyen autómatas, extensiones de estos para tratar variables, autómatas equipados con mecanismos de comunicación, etc.

## Conclusiones

En este capítulo se ha hecho la fundamentación teórica de los diferentes conceptos vinculados a los términos de Arquitectura de Software y Métodos formales, llegándose a la conclusión de que estos últimos permiten mejorar la especificación formal y hacerlas más claras y precisas, mediante notaciones matemáticas, que permiten que los errores se descubran rápidamente.

# CAPITULO II.TECNICAS DE ANALISIS DE PROPIEDADES ESTRUCTURALES

## Introducción

Los sistemas informáticos desempeñan un papel fundamental en la sociedad de la información, ellos son los encargados de controlar las redes para un mayor y rápido intercambio de la información a nivel mundial, permiten acceder a grandes volúmenes de información heterogénea y distribuida como cuentas bancarias, consultas bibliográficas y representan un medio no muy ajeno hoy día en las transacciones comerciales. También no se puede dejar de mencionar las ofertas y posibilidades que facilita Internet para el desarrollo de empresas, instituciones, y hasta en los hogares.

Esta gran dependencia a los sistemas informáticos hace inadmisible la ocurrencia de fallos donde se vean implicadas vidas humanas o cuantiosos daños materiales, sistemas de control de traco aéreo o terrestre, instrumentación médica, redes telefónicas o comercio electrónico. Diversas referencias bibliográficas se hacen eco de errores catastróficos que derivan en pérdidas irrecuperables, tal es el caso del fallido lanzamiento del cohete Arianne 5, con un costo de medio billón de dólares y la deficiente gestión informática de las olimpiadas de Atlanta en 1996.

Cualquier usuario habitual de un sistema informático puede constatar la presencia de anomalías y disfunciones en su equipo, que aceptará con más o menos resignación, aprendiendo con el tiempo, a convivir con estos errores, pero a medida que la sociedad se vincula con el sistema, se hace necesaria la necesidad de asegurar la corrección de su comportamiento.

Para alcanzar un elevado nivel de prestaciones que deben satisfacer los sistemas tecnológicos modernos, estos llegan a alcanzar una complejidad tal que su desarrollo, explotación y mantenimiento requieren la utilización, durante las distintas etapas de su vida útil, de técnicas y herramientas de asistencia.

Las técnicas y métodos automáticos, por naturaleza, descansan en fundamentos formales, que permiten al analista de requisitos o al programador hacer consultas en la aplicación. Esta perspectiva guía el uso de la lógica con fines computacionales. La lógica proporciona a la ingeniería el marco científico y tecnológico adecuado para construir una ingeniería creadora y sustentadora de la tecnología del software que la sociedad de la información necesita.

La tecnología resultante descansa en bases formales sólidas que permiten garantizar la corrección y efectividad de las técnicas desarrolladas. A diferencia de otros métodos formales, la lógica proporciona una aproximación ligera que integra armónicamente una variedad de lenguajes, herramientas y técnicas estándar.

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

## 2.1 Técnicas de análisis de propiedades estructurales

Se entiende por técnica a un procedimiento o grupo de procedimientos que tienen el fin de obtener un resultado específico sin importar el campo en donde se esté desenvolviendo. La definición de técnica manifiesta que ésta requiere de destrezas intelectuales como a su vez manuales, habitualmente para llevarla a cabo se necesita de la ayuda de herramientas y el adecuado conocimiento para manipularlas.

Las propiedades estructurales tratan de detectar para una solución específica aquellos bucles de control que no afectan al funcionamiento del sistema, debido a que no limitan el flujo de trabajos a lo largo de la línea. Por tanto las propiedades estructurales pueden ser empleadas de dos maneras diferentes:

- Para simplificar una cierta solución.
- Para generar soluciones estructuralmente eficientes.

En fin, las técnicas de análisis de propiedades estructurales, son un conjunto de procedimientos que se encargan de detectar el error, sin que este afecte el funcionamiento del sistema en la etapa del análisis.

Los métodos formales suelen entenderse como el uso de técnicas basadas en distintas teorías matemáticas. En el proceso de desarrollo de software se manejan habitualmente tres tipos de componentes: los programas (con la documentación asociada), las propiedades (especificaciones, tipos, restricciones temporales, requerimientos de corrección, prestaciones, entre otros) y los datos (selección de pares de entrada\salida, trazas, escenarios y ejemplos).

Los programas son documentos formales que deben someterse a un procesamiento automático como el que se realiza con herramientas como compiladores e intérpretes[7]. La sintaxis y semántica del lenguaje en que están escritos los programas determinan sus propiedades formales. Aunque la sintaxis suele estar mejor formalizada que la semántica (al menos en los lenguajes utilizados más comúnmente), la existencia de marcos formales potentes como la semántica denotacional, permiten dar soporte formal a herramientas automáticas como las que se aplican en el análisis, verificación y optimización mecánica de programas.

En un modelo ideal de desarrollo del software como es el ciclo de vida en cascada, los requisitos del sistema se determinan antes de comenzar el diseño de los programas. Estos también se pueden determinar al derivar la especificación, propiedades, requisitos, etc. a partir de los propios programas, mediante herramientas automáticas de inferencia de tipos o de análisis de programas, basadas, por ejemplo, en técnicas de interpretación abstracta, entre otras.

7 El intérprete en los lenguajes de programación simula una máquina virtual, donde el lenguaje de máquina es similar al lenguaje fuente, analiza directamente la descripción simbólica del programa fuente y realiza las instrucciones dadas.

### 2.1.1 Interpretación Abstracta.

La técnica de interpretación abstracta se basa en que la semántica de un lenguaje de programación puede ser más o menos precisa, según el nivel de observación elegido. Cuando se observa el programa sin exigir un alto nivel de detalle, se puede realizar una abstracción de su semántica que, aun siendo menos precisa que la original, en equilibrio es computable. La técnica de interpretación abstracta permite realizar analizadores de programas a base de definir un dominio de valores abstractos y una versión también abstracta del programa: la abstracción de los valores de datos es una descripción de propiedades que cumplen los datos, la versión abstracta del programa describe al programa. El análisis obtiene información sobre propiedades que cumple el programa a partir de simular la ejecución de la versión abstracta del programa con datos abstractos, imitando la ejecución del programa real sobre datos concretos. Se trata de utilizar esta técnica para realizar analizadores automáticos (programas que analizan otros programas) de programas lógicos.

Las pruebas realizadas a un sistema, plantean cierta dificultad para su realización: el potencial espacio de estados que un sistema de software puede alcanzar es enorme, y suele ser imposible recorrerlo por completo. Por lo que se hace necesario seleccionar un subconjunto significativo de dicho espacio de estados, apoyándose en criterios, en gran medida heurísticos. La automatización de este análisis resulta difícil. El análisis de programas de ordenador mediante otros programas de ordenador presenta dificultades especiales, e incluso límites absolutos de la computación. La interpretación abstracta es una aproximación a este problema.

En un modelo de desarrollo basado en el prototipo automático, las especificaciones pueden ser ejecutables; en este caso, se comportan a la vez como requisitos y como programas; permiten considerar la propia especificación como un programa, y en consecuencia, ejecutar de forma inmediata la especificación definida, lo que proporciona una forma simple de modelo, que permite al ingeniero de requisitos obtener una primera aproximación de los resultados que su especificación producirá en caso de ser correctamente implementada. De esta forma pueden desenmascararse deficiencias de planteamiento o comportamiento que

podría resultar muy costoso detectar y corregir en fases más avanzadas del proceso de producción de software. La utilidad del prototipo se ve reforzada por la disponibilidad de herramientas de generación y optimización automática, que aplican transformaciones basadas en la semántica, cuya corrección y efectividad están garantizadas formalmente. Esta forma de prototipo asistido contribuye a facilitar la generación de la especificación definitiva. En otros casos, las especificaciones pueden utilizarse como predicciones que permiten comprobar la corrección de los programas. Este es el caso de algunas técnicas empleadas en la depuración racional o algorítmica de programas.

Los datos pueden ser tanto de entrada o salida como datos internos que el programa puede utilizar o producir. Durante el desarrollo, muchas veces se utilizan trazas, que son secuencias de datos obtenidas en una ejecución del programa, usando un intérprete o monitor instruido específicamente para este propósito. Las trazas pueden existir también antes de los programas, como una forma de especificar su comportamiento y reciben el nombre de escenarios. Cuando los datos se usan como entrada de un proceso de aprendizaje, suelen denominarse ejemplos. En general, la corrección de las salidas respecto a las entradas se comprueba mediante el propio usuario si se hace manualmente o puede estar implementado mediante una especificación ejecutable.

### 2.1.2 Aprendizaje inductivo

El Aprendizaje Inductivo se basa en el aprendizaje a partir de la recopilación de datos, es decir, construye teorías a partir de los datos. Las técnicas de Aprendizaje Inductivo se basan en que el sistema pueda, automáticamente, conseguir los conocimientos necesarios a partir de ejemplos reales sobre la tarea que se desea modelar. En este segundo tipo, los ejemplos los constituyen aquellas partes de los sistemas basados en los modelos ocultos de Markov o en las redes neuronales artificiales, que son configuradas automáticamente a partir de muestras de aprendizaje. Su objetivo es descubrir descripciones generales de conceptos a partir de un número limitado de ejemplos (patrones comunes).

Los modelos de Markov describen un proceso de probabilidad, el cual produce una secuencia de objetos o símbolos observables. Son llamados ocultos porque hay un proceso de probabilidad subyacente, que no es observable, pero afecta la secuencia de eventos observables. Estos modelos fueron introducidos por Baum, el cual propuso un método estadístico de estimación de las funciones probabilísticas de una cadena de Markov. Sirven para modelar sistemas con tiempo dependiente del comportamiento caracterizado por procesos comunes de corta duración y la transición entre ellos. (9)

Los modelos de redes neuronales artificiales están compuestos de varios nodos simples operando en paralelo y arreglados en patrones, simulando redes neuronales biológicas. (9)

La inferencia inductiva es la base del aprendizaje inductivo. Es un conjunto de reglas heurísticas que permiten ir de lo específico a lo general. La mayoría de los métodos de aprendizaje las usan implícitas o explícitamente. En el proceso de inducción se elaboran conceptos generales que no contradigan lo que se conoce. Sus elementos lo constituyen un conjunto de premisas y un conocimiento de respaldo que puede ser un conocimiento específico o restricciones.

### 2.1.3 Técnica de propiedades a programas

En un proceso de desarrollo del software, las especificaciones anteceden a los programas. Siguiendo el principio conocido como derivación formal o síntesis de programas, el código de la aplicación puede ser obtenido a partir de la especificación del sistema. Sin embargo, pese a que los primeros trabajos sobre derivación formal de programas se remontan a los mismos orígenes de la programación como disciplina científica, estas técnicas han alcanzado una aceptación muy limitada, y los escasos ejemplos de aplicación práctica de este principio pasan por una automatización bastante parcial del proceso, como es el caso del Método B[8].

[8] B es el más reciente de dichos métodos. Fue diseñado por J.R. Abrial, quien participó también en la concepción de Z. La novedad de B es una notación estándar, la máquina abstracta, para especificar, diseñar e implementar sistemas.

### 2.1.4 Técnica de programas a propiedades

Son las técnicas más populares para extraer propiedades y especificaciones, a partir de los programas, incluyen el análisis estático, la inferencia de tipos y la verificación formal de programas. La técnica de la Interpretación Abstracta es una aproximación semántica, que se usa para proporcionar estáticamente en tiempo finito las respuestas correctas a cierto tipo de cuestiones relevantes sobre el comportamiento de los programas en tiempo de ejecución. Los datos y los operadores semánticos-concretos, se reemplazan por sus respectivas versiones abstractas. En este contexto, un análisis se ve como una computación aproximada, definida sobre descripciones de los datos en lugar de sobre los datos mismos. Diferentes estilos de definición semántica conducen a diferentes aproximaciones al análisis de programas. En el caso de los programas lógicos, existen dos aproximaciones principales: el análisis descendente (top down) y el análisis ascendente (bottom up). La aproximación descendente, que propaga la información en el mismo sentido que la regla de resolución, es quizás la más popular.

La principal diferencia entre ambas aproximaciones está en la independencia o no del análisis respecto al objetivo, siendo independiente en el caso ascendente y dependiente en el descendente. En otra línea se sitúan los análisis de prestaciones, que permiten estimar y mejorar la efectividad de las aplicaciones en cuanto a uso de recursos y rendimiento.

### 2.1.5 Técnica de programas a datos

Los métodos de prueba estructural o de caja blanca se basan en la siguiente estrategia: primero, se eligen determinados criterios de prueba, tales como el recorrido de un conjunto de caminos que exploran todos los usos de las definiciones; después, se construyen baterías de datos y se alimenta con ellos el programa en un entorno en el que se controla el cubrimiento de casos de acuerdo con el criterio considerado. El proceso se repite hasta que se alcanza un cubrimiento adecuado, que se mide como un porcentaje de las instrucciones, caminos posibles, etc. La tarea que representa el mayor desafío es la generación automática de los

juegos de datos. Una aproximación muy interesante consiste en seleccionar y especificar caminos que satisfacen un criterio de prueba en particular, y acumular las condiciones que definen dichos caminos mediante restricciones.

### 2.1.6 Técnica de datos a programas o a propiedades

La síntesis de programas a partir de ejemplos, también conocida como aprendizaje de programas, representa uno de los intentos más tempranos de mecanizar el desarrollo de programas. El marco más común para este tipo de métodos es la inferencia inductiva. No existe ninguna diferencia entre el aprendizaje de especificaciones y el aprendizaje de programas. En ambos casos, para hacer viable y efectivo el proceso, las funciones que se inducen pertenecen a fragmentos restringidos del lenguaje de especificación o de programación considerados.

### 2.1.7 Técnica de propiedades a datos

Las pruebas de programa son necesarias incluso si el desarrollo comienza con una especificación formal, y el desarrollo posterior es también completamente formal, puesto que la especificación podrá contener errores y en tal caso, la única forma de descubrirlos es comparar la especificación con otro tipo de propiedad o requisito formal. Puesto que no es conveniente esperar a validar el sistema final para encontrar el error, en ocasiones puede resultar útil validar la propia especificación. Esto se asemeja a las pruebas funcionales o de caja negra. Similarmente a las pruebas estructurales, se trabaja con una noción de cubrimiento que, en este caso, usa hipótesis u objetivos del test, escritas en el lenguaje de especificación considerado y que determinan de forma precisa los límites o alcances de las pruebas realizadas.

### 2.1.8 Técnica de propiedades a propiedades

Desde el momento en que las especificaciones pueden ser ejecutadas, no existe una frontera nítida entre las especificaciones y los programas. Algunos marcos formales proporcionan un

modelo para el refinamiento mecánico de especificaciones, que proceden de forma iterativa e incremental evolutiva hasta alcanzar una versión que se considera el programa. Si cada refinamiento preserva la semántica, entonces el proceso total produce un programa correcto acorde con las especificaciones iniciales. Generalmente, estos métodos están automatizados sólo parcialmente, como en el caso del Método B mencionado anteriormente.

### 2.1.9 Técnica de programas a programas

La compilación es la técnica más popular para producir programas a partir de otros programas. Normalmente, su salida se escribe en un lenguaje de menor nivel que el del programa de entrada y en el caso extremo, sólo resulta legible para la máquina. La compilación puede ser un proceso totalmente sintáctico durante el cual se traduce un programa en otro, pero también puede incorporar optimizaciones que exploten la semántica del lenguaje. También es posible transformar programas, sin cambiar de lenguaje, para extender su funcionalidad o para derivar versiones de mayor calidad, típicamente más eficientes. Esto último puede lograrse mediante técnicas de especialización y evaluación parcial de programas, que potencian el diseño de programas genéricos, que pueden usarse en varios contextos y después especializarse de forma mecánica para mejorar su eficiencia.

Existe otra forma de ir de datos (o programas) a otros programas mediante las técnicas de depuración racional, ya que cuando un programa no supera las pruebas, es posible usar los datos que demuestran que el comportamiento es incorrecto, que ayudará a localizar y corregir el error. Esto puede hacerse automáticamente, usando la especificación o una abstracción finita de su semántica que permita encontrar las partes del programa que son incorrectas mediante técnicas top-down, basadas en la exploración de árboles de computación, o usando técnicas bottom-up, que se basan en el operador de consecuencias inmediatas, como en los depuradores declarativos de los programas lógicos.

Para una mayor explicación referente al tema se debe explicar que las técnicas top-down consisten en tomar el problema en forma inicial como una cuestión global y descomponerlo sucesivamente en una solución más sencilla. Puede detenerse cuando los problemas

resultantes alcanzan un nivel de detalle que el programador o analista pueden implementar fácilmente. Sus objetivos principales son:

- Simplificación del problema y de los subprogramas de cada descomposición.
- Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.
- El programa final queda estructurado en forma de bloque o módulos lo que hace más sencilla su lectura y mantenimiento.

La técnica bottom-up es un diseño ascendente que se refiere a la identificación de aquellos procesos que necesitan computarizarse conforme vaya apareciendo su análisis como sistema y su codificación, o bien, la adquisición de paquetes de software para satisfacer el problema inmediato.

Puesto que estas técnicas usan como entrada tanto los casos de prueba (o datos) fallidos como una especificación del problema, y producen como salida un programa (el programa ya corregido), no pueden entenderse estrictamente como arcos que van de datos a programas sino más bien como híper-arcos que relacionan los programas, las propiedades y los datos. Como fase final del proceso, el programa puede corregirse mecánicamente, usando ejemplos que pueden generarse automáticamente a partir de la especificación, para inducir el código correcto.

### 2.1.10 Transformación de programas

Las técnicas de transformación de programas también desempeñan un papel primordial en la moderna tecnología de Model Checking, que permite verificar formalmente algunas propiedades de interés de los programas (en particular, de los programas reactivos: equidad, alcance, vivacidad). Para ello, primero se especifica formalmente la propiedad a analizar, por ejemplo, usando algún tipo de lógica temporal, como la lógica CTL o Lógica Temporal Ramificada.

La lógica temporal posee la 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 permite trabajar de forma directa con especificaciones parciales. La relación de satisfacción que se define en el campo de la lógica admite proceder a la verificación de propiedades en una parte del sistema, sin necesidad de tratar con la especificación completa.

La técnica de transformación de programas viene dada ya que dado un programa X, se puede generar un programa X' que resuelve el mismo problema y es semánticamente equivalente a X, pero que posee mejor comportamiento respecto a cierto criterio de evaluación. Un ejemplo de esto lo constituyen los compiladores, ya que estos transforman un programa fuente en un programa objeto preservando su semántica con el objetivo de ganar eficiencia en la ejecución.

La técnica de transformación de programas es fundamental en la derivación de programas funcionales y de creciente importancia en los otros paradigmas. Esta técnica está incorporada al conocimiento básico en desarrollos de programas e ingeniería de software.

La Evaluación Parcial es una técnica de transformación automática de programas que persigue, entre otros objetivos, la optimización de programas con respecto a ciertos datos de entrada; de ahí que también haya recibido el nombre de especialización de programas. Es una técnica de transformación de programas fuente a fuente, en la cual, dado un programa P y conociendo parte de los datos de entrada, se pre-computan aquellas porciones de P que sólo dependen de los datos conocidos de entrada. El programa transformado resultante (también conocido como programa residual) es menos general que el programa original, pero puede ser mucho más eficiente. Los datos de entrada que son conocidos de antemano reciben el nombre de datos estáticos, mientras que el resto de los datos de entrada se conocen como datos dinámicos.

### 2.1.11 Técnica de Model Checking:

La técnica de Model Checking constituye una verificación formal y automática y se aplica a sistemas con un número finito de estados. Dado un modelo de sistemas y la propiedad requerida, permite decidir automáticamente si la propiedad es satisfecha por este modelo o no. (10)

En comparación con las técnicas de demostración de teoremas, que se aplican también a la verificación formal de programas, y que consisten en aplicar métodos computacionales para demostrar teoremas, es decir, demostración de teoremas con un ordenador. La ventaja del Model Checking es que se aplica de forma totalmente automática y sin requerir una habilidad especial en lógica matemática por parte del usuario.

De forma análoga a las técnicas de depuración racional de código y la verificación algorítmica de programas toma varios tipos de entradas, en particular, el programa y la propiedad a verificar y produce varias salidas: la especificación de los estados iniciales que verifican la propiedad o el contraejemplo que demuestra que esta no es cierta.

Los métodos y técnicas de la lógica pueden aplicarse prácticamente en todas las etapas del desarrollo de proyectos. Es bien conocida la dificultad de realizar una especificación completa inicial del problema a resolver sin obviar detalles relevantes para el cliente o para el personal implicado en el desarrollo. Las limitaciones humanas, tanto de previsión, expresión y comunicación, como de comprensión, se unen para hacer del desarrollo de proyectos un proceso iterativo donde cada vuelta atrás resulta costosa, repercutiendo cada nuevo cambio introducido en un coste económico y temporal muy acentuado. Este problema está revelando su importancia también en los aspectos de gestión, hasta el punto que el mismo tipo de recursos lógicos que pueden usarse durante el proceso de desarrollo se han propuesto también para las actividades de gestión como por ejemplo, para la especificación formal de los contratos de software y también para la ayuda integrada a la toma de decisiones.

Las técnicas de verificación algorítmica (11) o Model Checking tienen un triple atractivo ya que:

- Son completamente automáticas.
- Su aplicación no requiere supervisión por parte del usuario ni experiencia en disciplinas matemáticas (contrariamente a las técnicas de verificación declarativa más clásicas, que se basaban en el uso experimentado, por parte del usuario, de sistemas de axiomas y reglas de inferencia), y en el caso de no verificarse la propiedad analizada, la herramienta produce un contraejemplo que ayuda a identificar la fuente del error.

**Herramienta SPIN**

El SPIN es una herramienta utilizada para especificar algoritmos, los pasos que sigue son los siguientes:

- Se construye el programa.
- SPIN traduce el programa a un autómata finito.
- Se expresa la propiedad a verificar, usando lógica temporal.
- SPIN traduce la propiedad a un nuevo autómata.

SPIN verificará que el lenguaje del autómata del programa está incluido en el lenguaje del autómata de la propiedad a verificar, si es así, la propiedad se cumple, en caso contrario, SPIN mostrará la secuencia de líneas de código que llevarán al error.

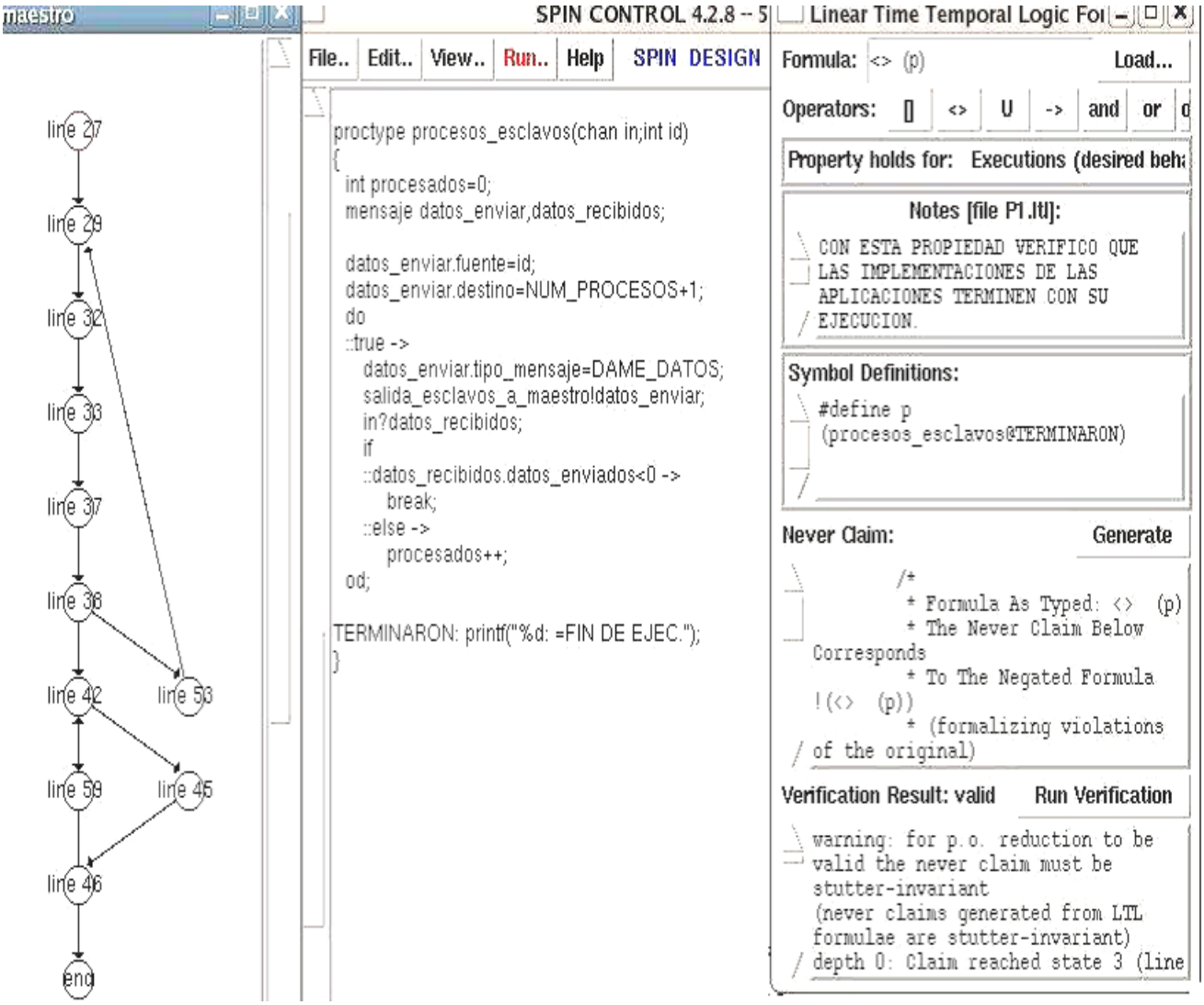


Fig.1 Interfaz gráfica de SPIN, donde el proceso ocurre de manera automática

### 2.1.12 Técnicas Declarativas

Las técnicas declarativas son particularmente de gran alcance en la Interfaz Utilizador y la Interfaz de Programación de Aplicaciones (APIs) que tienen un rico complejo establecido de entradas sobre un campo relativamente pequeño de los comportamientos de la ejecución. Dos ejemplos del software comercial que ilustran la aplicación de técnicas declarativas son DriverLINX y ExceLINX en los campos del control del instrumento de la adquisición y de la prueba de datos:

- DriverLINX es un APIs para controlar el hardware de adquisición de datos, usado para medir y para generar las señales análogas y numéricas interconectadas a todos los tipos de transductores[9] externos. Los usos de adquisición de datos incluyen la investigación del laboratorio, la instrumentación médica, y el control del proceso industrial.
- ExceLINX es un dispositivo suplementario al Microsoft Excel que permite la especificación rápida de las disposiciones de prueba usando las formas de la hoja de trabajo. Los usuarios especifican o declaran los canales y configuraciones, muestreando tarifas, accionando, y localizando los datos para las medidas que desean realizarse completando una hoja de trabajo del Excel. Cuando el usuario selecciona el botón de "comienzo" en la barra de herramientas, ExceLINX traduce la especificación a la secuencia correcta del comando, inicia la medida, y fluyen los datos de nuevo a la hoja de trabajo solicitada. Los usuarios pueden dar"setup" y recoger medidas por sí mismos en minutos, usando las especificaciones de la lógica, comparadas a los días o a las semanas usando el tiempo del programador para las especificaciones imprescindibles. (12)

Estas ventajas y el uso de técnicas simbólicas, que permiten la enumeración explícita de un número astronómico de estados, han revolucionado el campo de la verificación formal transformándolo de disciplina académica en tecnología de amplio uso industrial. El uso de algún tipo de lógica modal permite verificar propiedades dinámicas de los sistemas. Un ejemplo que demuestra la potencia de estas técnicas es su aplicación a la verificación del protocolo de coherencia de cache descrito en el estándar IEEE Futurebus+[10] (IEEE Standard 896-1-1991), que permitió localizar errores ya conocidos pero aun no identificados, así como diagnosticar otros previamente no detectados. Por otro lado, en un contexto abierto de programación, a través de Internet, resulta conveniente interponer barreras de acceso

[9] Un transductor es un dispositivo capaz de transformar o convertir un determinado tipo de energía de entrada en otra diferente de salida.

[10] Futurebus+ es un estándar de bus asíncrono de altas prestaciones desarrollado por IEEE. Futurebus+ es una especificación de bus compleja. Introduce algunos conceptos novedosos en el diseño de buses.

seguras, basadas en las técnicas formales de certificación de código que es posible reinterpretar en términos de aproximación de propiedades y verificación de dichas aproximaciones. El proceso de certificación se basa entonces en un análisis estático del código para determinar si este satisface una determinada política de accesos y flujos de información, especificada por las clases de información que pueden albergar las variables del programa y las relaciones legales entre diferentes clases.

La certificación de código garantiza y demuestra en un nivel formal que el software posee condiciones de calidad. Esto consiste en requerir al productor de código la evidencia necesaria, por ejemplo una prueba, de que su código satisface las propiedades deseadas. Esta evidencia sirve como un certificado que puede ser verificado independientemente. Requiere muchas anotaciones detalladas para hacer posible la prueba. Estas anotaciones resultan engorrosas generarlas manualmente, por lo que existe una herramienta que acelera y formaliza el proceso de certificación: la compilación certificante.

El esquema para producir software confiable mientras se compila puede ser dividido en dos pasos. El primer paso consiste en compilar el código fuente e introducir anotaciones en el archivo objeto (bytecode para JAVA, assembly para C, o cualquier otro lenguaje intermedio).

En el segundo paso, el código anotado es tomado por un verificador que, basado en una especificación formal de seguridad, chequea que el código anotado no viole las condiciones previamente establecidas. Si la verificación tuvo éxito, la ejecución de la aplicación final será segura.

Un compilador certificante usa información estática de tipos para generar código seguro, traduce el programa y la información de tipos en un lenguaje intermedio anotado. Las certificaciones realizadas por dicho compilador pueden ser usadas para generar optimizaciones sobre el código, y las verificaciones realizadas sobre estas certificaciones son de gran utilidad para depurar modificaciones y extensiones del compilador. (13)

La certificación automática combina las técnicas de análisis estático con la verificación algorítmica y la depuración racional del código. Es fundamental, por tanto, disponer de lenguajes de alto nivel que permitan especificar problemas, propiedades, y algoritmos eficientes para resolver dichos problemas y analizar las propiedades de interés. La dualidad entre lenguajes lógicos y clases de autómatas surge como una particularización de esta dualidad entre lenguajes de especificación y algoritmos, que al menos, para cierta clase de problema, satisface los requisitos deseables por el programador.

Los lenguajes lógicos utilizan un conjunto bastante reducido de recursos sintácticos: conectivas, cuantificadores, igualdad y fórmulas atómicas específicas, además de términos y variables que son interpretados en dominios concretos. Los autómatas son algoritmos que pueden ser manipulados, combinados, descompuestos, compilados, optimizados, reducidos, etc. La dualidad entre operadores lógicos y algebraicos facilita el desarrollo modular de las especificaciones. Esta dualidad ha sido utilizada con éxito para desarrollar diversas técnicas de verificación de propiedades en muy distintos campos tecnológicos. La conexión entre lenguajes lógicos y autómatas se remonta a los trabajos de Buchi y Elgot, quienes demostraron que las transformaciones de fórmulas a autómatas, y viceversa, son efectivas. También fue posible demostrar que las teorías monádicas de segundo orden son decidibles. Esto tiene una aplicación directa si se es capaz de enunciar una propiedad sobre un sistema de software determinado empleando una fórmula de la lógica monádica de segundo orden. Recientemente, los sistemas concurrentes sujetos a cambios dinámicos han cobrado nueva actualidad en el contexto del Trabajo Cooperativo Asistido por Ordenador (CSCW). Un punto de partida habitual son las redes de Petri, formalismo matemático que permite modelar, analizar, simular, controlar y evaluar el comportamiento de sistemas concurrentes y distribuidos. Con una red de Petri pueden estudiarse dos tipos de propiedades:

- Las que dependen del marcado inicial (propiedades de comportamiento: alcance, acotamiento, vivacidad, reversibilidad, cobertura y persistencia).
- Las que son independientes del marcado inicial (propiedades estructurales: acotamiento, estructura, control, conservación, repetición y consistencia).

Dentro del campo CSCW, las redes de Petri han jugado un papel importante en el desarrollo de los sistemas Workow, sistemas que ayudan a las organizaciones a especificar, ejecutar, monitorizar y coordinar el flujo de tareas dentro de un sistema distribuido. Dado que el formalismo de las redes de Petri no ofrece una manera directa de representar características tales como cambios dinámicos de actividades, migración de tareas, modos de operación múltiples, etc.; para modelar los sistemas Workow se utilizan extensiones de las redes de Petri diseñadas especialmente para permitir el modelado de alguna de estas características, como las redes reconfigurables, que permiten el cambio dinámico en la propia estructura de la red, y los autómatas cooperantes, en los que un sistema puede verse como una red de Petri de mayor nivel que el tradicional en la que las marcas, son elementos activos que consisten en un autómata junto con una porción de memoria privada:

- Las redes reconfigurables tienen su origen en dos líneas de trabajo diferentes, ambas relacionadas con el formalismo de las redes de Petri y cuyo propósito es mejorar la expresividad del modelo básico de las redes de Petri para soportar la descripción de sistemas concurrentes sujetos a cambios dinámicos. La primera línea de trabajo estudia la manera de fusionar redes de Petri con gramáticas de grafos, mientras que la segunda, representada en particular por las redes automodicantes de R. Valk, estudia las redes de Petri cuyas relaciones de flujo pueden cambiar en tiempo de ejecución.
- En el formalismo de los autómatas cooperantes, las transiciones de la red son principalmente vectores de sincronización que dan todas las posibles interacciones entre objetos dinámicos. Se asume, básicamente, que los objetos pueden sincronizarse de acuerdo con su comportamiento (el servicio que prestan o que solicitan) independientemente de su estado interno.

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.

En estas extensiones, lo que se gana en poder de modelado se traduce en pérdida de claridad de algunas propiedades de interés. El desafío está, por tanto, en conseguir un equilibrio entre expresividad y computación. Es importante que el mecanismo que maneje el cambio dinámico esté representado explícitamente en el modelo para que, en cada etapa del proceso de desarrollo, los diseñadores puedan experimentar el efecto de los cambios estructurales, por ejemplo, utilizando prototipos. Esto significa que los cambios estructurales se tienen en cuenta desde el inicio del proceso de diseño en lugar de ser tratados por un sistema global y externo.

### 2.1.13 Técnica Proof-Carrying Code (Prueba Portadora de Código)

Técnica que garantiza el código móvil seguro. Requiere tanto la construcción de pruebas matemáticas de propiedades de programas que sean fácilmente verificables como de la especificación formal de propiedades de seguridad. Exige a un productor de software proveer su programa conjuntamente con la prueba formal de su seguridad. La política de seguridad del destinatario se formaliza mediante un sistema de axiomas y reglas de inferencia, sobre el cual debe basarse la demostración construida por el productor. El consumidor, por su parte, verifica que la prueba sea válida y que el código recibido corresponda a la demostración, y sólo ejecuta el código en caso de que ambas respuestas sean positivas.

La técnica de prueba portadora de código, no requiere autenticación del productor, ya que el programa sólo correrá si localmente se ha demostrado su seguridad, minimizando así la cantidad de confiabilidad externa requerida. Tampoco se requiere verificación dinámica, con lo cual no se introduce ningún deterioro en el tiempo de ejecución, ya que la prueba se efectúa con antelación.

## 2.2 Solución propuesta

A partir del análisis y caracterización realizado anteriormente sobre las técnicas de análisis de propiedades estructurales se considera que la propuesta de las técnicas esté conformada por la Técnica de Verificación Algorítmica o Model Checking. La siguiente propuesta permitirá que el software gane en eficiencia y cumpla los parámetros impuestos por el cliente.

## 2.3 ¿Por qué se excluyeron otras técnicas?

A raíz del estudio y la investigación realizada, de la recopilación de artículos, libros y documentos, se ha comprobado, que a pesar de que las técnicas de análisis de propiedades estructurales existen, aún el campo de aplicación donde se les conoce es muy limitado. Mucho de esto se debe al desconocimiento que existe vinculado a las técnicas de análisis. Otro factor que influye, podría decirse, ha sido la poca capacitación que presentan muchos de los desarrolladores de software en el campo de las matemáticas, lo que ha traído como resultado que muchos opten por la vía aparentemente más fácil, las técnicas no formales.

Se excluyeron aquellas técnicas que:

- Las herramientas que utilizan los proyectos no son capaces de soportarlas.
- No están completamente enfocadas a la detección de errores, sino a la certificación de la corrección de un sistema.

## 2.4 ¿Por qué utilizar la Técnica de Verificación Algorítmica o Model Checking?

La Técnica de Verificación Algorítmica o Model Checking, como se planteó en el capítulo anterior es completamente automática y no requiere supervisión del usuario. La técnica de Model Checking la herramienta que emplea es el SPIN, utilizada para especificar los algoritmos usados.

En la UCI la herramienta utilizada es la Máquina Virtual Paralela (en lo adelante PVM), parecida al SPIN, que ofrece un conjunto de primitivas para el intercambio de mensajes, lo cual permite el desarrollo de algoritmos paralelos. Esta herramienta es empleada para el trabajo de algoritmos complejos como es el cálculo de matrices.

PVM brinda la posibilidad de un mayor control de procesos y de una abstracción mayor en la manipulación de los mensajes. Por otra parte la forma en que ejecuta los procesos permite de forma implícita el desarrollo de módulos de procesamiento bien definidos. PVM cuenta con dos programas principales, el máster y el esclavo, que tienen básicamente la siguiente sintaxis:

Programa máster:

Paso 1: Conectarse a PVM

Paso 2: Crear procesos esclavos

Paso 3: Enviar datos

Paso 4: Procesar datos

Paso 5: Recibir resultados

Paso 6: Conformar solución final

Paso 7: Emitir la solución final

Programa esclavo:

Paso 1: Conectarse a PVM

Paso 2: Recibir datos

Paso 3: Procesar datos

Paso 4: Enviar resultados

La diferencia fundamental radica en que el código del máster está separado y compilado del esclavo y cada máster puede ser compilado para ejecutar una cantidad x de procesos esclavos y cada esclavo a su vez puede ser máster de otro conjunto, esto permite entre otras cosas desarrollar proyectos de una forma más modular y extensible y el procesamiento de grandes volúmenes de datos en tiempos increíblemente rápidos. Lo que normalmente duraría semanas ahora se tardaría apenas unas milésimas de segundos. Esto funciona como un clúster, es decir, un conjunto de computadoras interconectadas con dispositivos de alta velocidad que actúan en conjunto usando el poder de cómputo de varios CPU en combinación, para resolver ciertos problemas dados.

PVM actúa como un conjunto de programas y librerías que convierte a varias máquinas heterogéneas en un único multicomputador o máquina virtual. Sus ventajas radican en la portabilidad, pues los programas son, en lo que a PVM se refiere, portables, y en la diversidad de máquinas que se pueden utilizar.

## 2.5 Alcance de la investigación

La calidad del software es una de las problemáticas más importantes en los procesos de desarrollo de software. Garantizar el correcto funcionamiento bajo situaciones no determinadas es una tarea que tiene que ser realizada con cuidado extremo. En algunos casos, este tipo de pruebas son de mayor importancia, ya que involucran ambientes sensibles e información crítica en donde es necesario garantizar que cada uno de los componentes involucrados (hardware, software y componentes humanos) actúe de manera correcta ante situaciones específicas, con una variedad de ejemplos que cubren áreas tan diversas como planeación de tráfico, aplicaciones militares, sistemas médicos, y muchas otras.

Las técnicas de análisis de propiedades estructurales contribuyen a la construcción de software fiable. A medida que estas técnicas se vayan perfeccionando e implementando en los distintos proyectos productivos, se optimizará el código y se eliminarán los errores en las sentencias de detección, lo que impedirá la detección tardía de fallos en los sistemas que provoquen confusiones, pérdidas humanas y materiales.

En la actualidad, existe una tendencia al crecimiento de las prestaciones de las aplicaciones que necesitan procesar grandes volúmenes de datos en tiempos moderados, un ejemplo de esto lo constituye el trabajo del profesor Edisel Navas: “Reducción del Efecto de Multiscattering en la Medición del tamaño de Partículas por Extinción de la luz”.

El trabajo con matrices donde se encuentran implicados un número infinito de datos para el cálculo, puede ser muy útil en la realización de módulos de predicción económica o el filtrado de datos para analizar el comportamiento de los recursos empleados en las actividades de las FAR; también permitiría desarrollar módulos biotecnológicos, procesamiento meteorológico, experimentos físicos, módulos de inteligencia artificial en sus diferentes formas, entre otros.

El desarrollo de las matemáticas ha traído grandes progresos en la humanidad, a raíz de la aplicación de la mecánica celeste, la electricidad, los sectores de las ciencias naturales y técnicas.

### 2.6 Vinculación de la Técnica de Verificación Algorítmica o Model Checking en los proyectos productivos de la UCI

El proyecto productivo "El Frameworks APPD" de la facultad 4, en estos momentos se encuentra empleando la herramienta PVM, que utiliza la Técnica de Verificación Algorítmica o Model Checking para lograr:

- Alta disponibilidad en la infraestructura y en la aplicación, que consiste en conformar utilizando la Técnica de Verificación Algorítmica, un sistema que permita disponibilidad permanente de determinado servicio, minimizando las fallas del mismo.
- Alto rendimiento, que facilite la realización mediante la técnica propuesta la ejecución de determinados algoritmos que usualmente son imposibles de realizar o su tiempo de realización es muy elevado, en un plazo relativamente corto.

## Conclusiones:

Este capítulo se ha encaminado a abordar las características de las técnicas de análisis de propiedades estructurales estudiadas, que han contribuido a elaborar la propuestas que el autor considera sea más factible para el logro de un buen proceso de software, la solución antes expuesta, permitirá que el software gane en, calidad y eficiencia, con la especificación y verificación del código de un programa y beneficiará al programador con los cálculos de algoritmos que mostrarán las correcciones existentes, evitando posibles incongruencias.

# CONCLUSIONES

Obtener un producto de software sin errores es la meta que persigue el cliente y el desarrollador del sistema. Las técnicas de análisis de propiedades estructurales ha brindado la posibilidad de obtener un producto que satisfaga la calidad y la eficiencia deseada. Al realizar una detección temprana de errores propician la creación de sistemas que sean más fiables y seguros permitiendo verificar la implementación y transformar el software.

A raíz del análisis realizado en los proyectos productivos y la problemática encontrada, se llevó a cabo una exhaustiva búsqueda bibliográfica sobre las técnicas de análisis de propiedades estructurales, para el desarrollo del presente Trabajo de diploma, que permitió la elaboración de la propuesta a utilizar, la Técnica de Verificación Algorítmica o Model Checking, con el alcance de obtener un producto de software confiable y seguro para darle cumplimiento al objetivo planteado en la investigación.

Este trabajo ha brindado la posibilidad de ampliar los conocimientos hacia un área poco conocida de la Arquitectura de Software, pero no menos interesante que empieza desde hoy a dar sus primeros pasos.

## RECOMENDACIONES:

- Se recomienda seguir con el estudio de las técnicas de análisis de propiedades estructurales, ya que pueden ser muy útiles en el desarrollo de los proyectos productivos, así como ampliar el uso de las distintas herramientas que estas utilizan.
- Implementar el uso de las técnicas de análisis de propiedades estructurales a todas las facultades de la universidad.

# BIBLIOGRAFÍA

## Bibliografía Citada:

1. **Leslie Cruz Rodríguez, Isak J. Rodríguez Armenteros** http://www.somosjovenes.cu/index/semana76/informatic.htm.Cuba.Abril,2007.

2. **Pressman, Roger.** "*La Ingeniería del Software. Un enfoque practico".6ta Edición.2006.*

3. **Mara Alpuente, Salvador Lucas.** "*Introducción a la Ingeniería del Software Automática.* Valencia.2003".

4. **Ivar Jacobson, Grady Booch,James Rumbaugh.** "*El Proceso Unificado de Desarrollo de Software".*

5. **John C.Georgas, Eric M. Dashofy,Richard N. Taylor.** "*Desarrollo Centrado en la Arquitectura:Un acercamiento diferente a la Ingeniería de Software".* 2004.

6. **Reynoso, Carlos Billy.** "*Introducción a la Arquitectura de Software.Buenos Aires".* Marzo,2004.

7. **Canal, Carlos.** "*Un lenguaje para la especificación y validación de Arquitectura de Software".Diciembre,2000.*

8. **Larman, Craig.** "*UML y Patrones". 2da Edicion.2003.*

9. **Fragoso, Clemente.** "*Entrenamiento y comparación de un nuevo reconocedor de propósito general basado en redes neuronales y modelos ocultos de Markov".Mayo,2001.*

10. **Dr.P.R.D'Argenio.** "Técnicas Formales para la Verificación y el Desarrollo de Programas Reactivos". 2006.

11. **J.L. Ruiz-Reina, J.A. Alonso, M.J. Hidalgo y F.J. Martín-Mateos.** "*Verificación formal y eficiencia:Un caso de estudio aplicado a la unificación de términos" . España.2002.*

12. **Furman, Roy.** "*Programación declarativa:estrategias para solucionar problemas de programación".* Noviembre 12,2005.

13. **Francisco Bavera, Martin Nordio,Jorge Aguirre,Gabriel Baum,Ricardo Medel."***Desarrollo de un prototipo de compilador Certificante".Uruguay.Noviembre,2003.*

## Bibliografía Consultada:

**http**://www.eads.com/1024/es/eads/history/airhist/1950_1959/Lockheed_F-104_G.html. [En línea] .Mayo,2008.

**Sáez,José**."Grandes errores en la historia de la informática.Universidad de Murcia.Diciembre",2006.

**Faustino Gimena, Pedro Gonzaga,Lazaro Gimena**."Análisis estructural sistemático.Teorías,técnicas y aplicaciones.Su mapa conceptual como herramienta didáctica y de investigación". 2004.

**http**://www.daedalus.es/inteligencia-de-negocio/sistemas-complejos/ingeniería-de-sistemas/análisis-de-sistemas/

**Rolando A. Hernández León, Zayda Coello González**. "El Paradigma Cuantitativo de la Investigación Científica". Cuba. Noviembre, 2002.

**Vaucheret, Claudio**."Análisis Estático de Programas para la Generación de Computación Ubicua: Inferencia de Tipos".2007.

**Edisel Navas, Mayra P. Hernández."** *Reducción del Efecto de Multiscattering en la Medición del Tamaño de Partículas por Extinción de la luz".M*arzo,2007.

**M. Aguilar-Cornejo, J. L. Quiroz-Fabián, G. Román-Alonso y J. R. Jiménez-Alaniz."***Verificación formal de un algoritmo de procesamiento paralelo de imágenes médicas".J*unio,2007.

# GLOSARIO DE TERMINOS

**Programas reactivos:** Los programas reactivos son aquellos que no tienen estados terminales definidos y su ejecución consiste en una interacción continua con el medio ambiente (son programas que no terminan nunca).

**Reglas heurísticas:** Las reglas heurísticas son aquellas reglas que con base en un conocimiento previo indican que acción tomar. Se denomina heurística a la capacidad de un sistema para realizar de forma inmediata innovaciones positivas para sus fines.

**Wright**: La notación Wright permite realizar análisis estáticos acerca de la compatibilidad de las conexiones y la ausencia de bloqueos, pero no permite la creación dinámica de procesos, es decir, analiza la compatibilidad de las conexiones entre componentes y roles.

**Programación Extrema(XP):** Es una metodología de desarrollo ligera, que persigue el objetivo de aumentar la productividad a la hora de desarrollar programas.

**Sistemas de Misión Críticos:** Son los que proporcionan la estructura vital de funcionamiento y las transacciones involucradas en realizar los movimientos de los números de cuentas. Estos sistemas pueden ser un grupo grande, una serie de servidores o una cúpula de almacenamiento de datos. Los sistemas de Misión Crítica deben convertirse y rodearse de una arquitectura de recuperación, estar protegidos contra riesgos y ser operados en un esquema al margen de los usuarios finales.

**Peer-to-peer:** La red peer-to-peer o de punto a punto hace referencia a una red que no tiene clientes ni servidores fijos, sino, una serie de nodos que se comportan simultáneamente como clientes y servidores de los demás nodos de la red.

**Script**: Es un guión o un conjunto de instrucciones, que permite la automatización de tareas creando pequeñas utilidades. Usualmente son archivos de texto. También se considera como un script una alteración o acción a una determinada plataforma.

**Intérprete:** En los lenguajes de programación simula una máquina virtual, donde el lenguaje de máquina es similar al lenguaje fuente, analiza directamente la descripción simbólica del programa fuente y realiza las instrucciones dadas.

**Futurebus+:** Es un estándar de bus asíncrono de altas prestaciones desarrollado por IEEE, Instituto de Ingenieros Eléctricos y Electrónicos (por sus siglas en ingles Institute of Electrical and Electronics Engineers). Futurebus+ es una especificación de bus compleja. Introduce algunos conceptos novedosos en el diseño de buses.

**Lógica CTL ó Lógica Computacional en Árbol:** Especifica propiedades de un árbol de un sistema de transición. Este árbol infinito se obtiene designando a un estado del sistema como la raíz (estado inicial), y muestra todas las posibles computaciones desde ese estado.

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

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

**Axioma:** Leyes de la matemática que no requieren demostración.