UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE INGENIERÍA

MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA

"Arquitectura de un Simulador de Seguimiento Temporal de Calendario de Proyectos de Software"

TESIS

QUE PARA OBTENER EL GRADO DE
Maestro en Ingeniería

PRESENTA
Miguel Ángel Morales Almada

DIRECTOR
M. C. José Martín Olguín Espinoza

Mexicali, B. C. 09 de noviembre de 2007

RESUMEN

Administrar proyectos de software es una tarea compleja, en la cual intervienen numerosas variables que pueden afectar las fechas de entrega del proyecto. El administrador de proyectos de software debe estar en una constante búsqueda de métodos y modelos que lo ayuden a disminuir la incertidumbre en el desarrollo de software. El presente trabajo describe la propuesta para la creación de una arquitectura flexible de un simulador de seguimiento temporal de calendario de proyectos de software, tomando en cuenta los modelos de simulación, métodos de administración de actividades y modelos para la gestión de riesgos. La arquitectura propuesta consta de siete componentes (Componente de Red, Componente de Riesgos, Componente Historial, Componente Simulador, Controlador de Componentes, Manejador de Persistencia e Interfaz de Usuario), que se podrán intercambiar por otros de acuerdo a las necesidades del administrador de proyectos.

ABSTRACT

Managing software projects is a complex task in which numerous variables can affect the delivery date of the project. The project manager must constant search for methods and models that help diminish the uncertainty in the software development. This work describes a proposal for the construction of an architecture for a simulator that models the temporal flow of software project calendars, using modelling, program management and models for risk management, the proposed architecture consists of seven components (Network Component, Risk Component, History Component, Simulation Component, Component Controller, Persistence Manager, and User Interface), that will be interchangeable according to the project manager's requirements.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

CAPÍTULO I. INTRODUCCIÓN

La administración de proyectos es la aplicación de conocimientos, habilidades, técnicas y herramientas a las actividades de un proyecto, con el fin de satisfacer, cumplir y superar las necesidades y expectativas de los involucrados [Yamal, 2002]. Administrar proyectos de software es una tarea compleja en la cual intervienen variables que pueden afectar las fechas de entrega [Hamid, 1997]. Los administradores de proyectos requieren herramientas que les permitan planear las actividades y administrar los riesgos bajo un ambiente simulado en el que puedan representar proyectos reales con sus dificultades.

Los proyectos de software se asocian comúnmente con tiempos y presupuestos que rebasan las estimaciones originales. Una inspección de Standish Group, marcó las expectativas realistas como uno de los cinco factores principales necesarios para asegurar el éxito de los proyectos internos de software de gestión [Standish Group, 1994].

En la Figura 1.1, se muestra el porcentaje de éxito obtenido en los proyectos internos [Standish Group, 1994], los tipos se describen a continuación**:**

Tipo 1.- Proyecto exitoso. El proyecto es terminado a tiempo y en presupuesto estimado, con todas las características y funciones según lo especificado inicialmente.

Tipo 2.- Proyecto comprometido. El proyecto es terminado, pero sobrepasando las estimaciones originales de tiempo y presupuesto. Además, ofrece menos de las características y funciones que fueron especificadas originalmente.

Tipo 3.- Proyecto fallido. El proyecto se cancela en un cierto punto durante el ciclo de desarrollo.

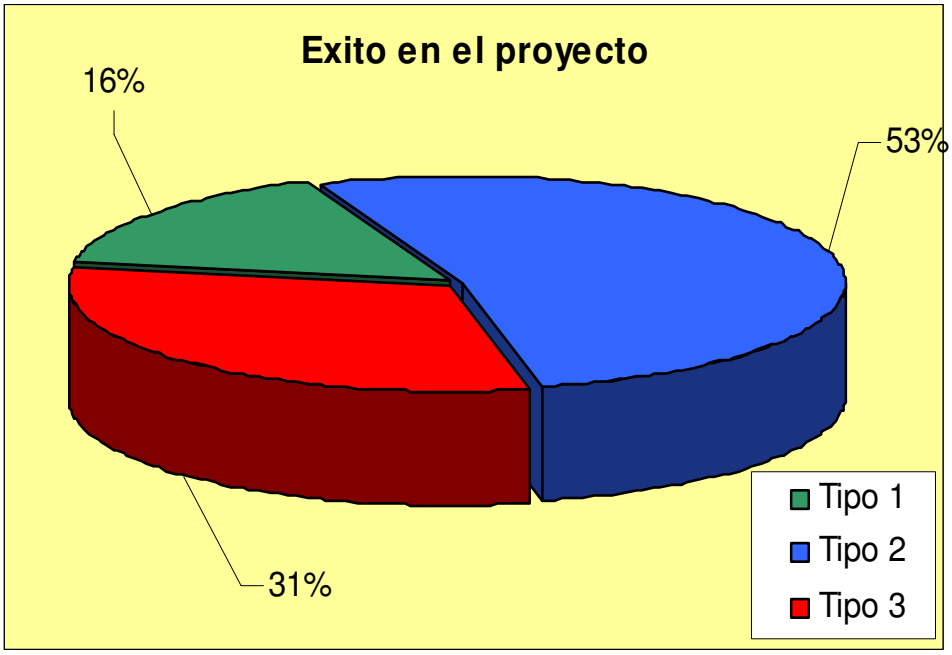

Figura 1.1. Éxito obtenido en los proyectos,

Los proyectos de software con calendario acotado tienden al fallo por dos causas: la inadecuada asignación de prioridad de los requisitos y la incapacidad que tienen los métodos tradicionales de planeación para manejar la incertidumbre de las estimaciones sobre las que se basan los planes de

trabajo; incertidumbre asociada al hecho de que el desarrollo de software no progresa en forma lineal [Miranda, 2002].

El administrador de proyectos debe tomar en cuenta diversos factores que influyen directa o indirectamente en la fecha de entrega de su producto: cambio de los requerimientos, falta de capacitación, rotación de personal, etc. Esto sin incluir que las técnicas y métodos que se utilizan para la construcción del software varían siempre de un proyecto a otro, no importando que tan similares puedan ser [Hamid, 1997].

El presente trabajo se enfoca en el desarrollo de una arquitectura flexible de un simulador de seguimiento temporal del calendario de proyectos de software, que tenga componentes intercambiables, basada en el hecho de que existen múltiples métodos para el seguimiento temporal de proyectos y para el análisis de los factores de riesgos.

1.1 Antecedentes

La adecuada programación del proyecto es una tarea esencial y extremadamente difícil en la administración del software. El tiempo necesario para terminar una actividad del desarrollo a menudo se retrasa. La simulación se puede utilizar para apoyar a administradores del software en encontrar los calendarios óptimos para sus proyectos [Padberg, 2003].

Padberg (2003) presentó un modelo discreto para la simulación de proyectos de software. Este modelo representa asignaciones de tareas, niveles de habilidad del personal, componentes. Estudió sistemáticamente el funcionamiento de varias políticas para un proyecto muestra. También realizó un análisis detallado de las asignaciones de tareas que se presentan en las simulaciones, estableciendo que el método de simulación es una técnica importante para evaluar la calendarización de actividades y el impacto de los cambios de las actividades en el proyecto.

Miller y Lee (2004) realizaron un estudio para tratar de integrar el método de sistemas dinámicos con la red multi-proyectos utilizando el método de la cadena crítica (secuencia de eventos dependientes que evitan que el proyecto se complete en un intervalo más corto de tiempo) en un ambiente simulado.

El método de la cadena crítica se explica a detalle en el capítulo 2, el modelo de administración de multi-proyectos propuesto por Miller y Lee, se usó para diferentes pruebas de impacto en el comportamiento de proyectos múltiples.

Las razones por las cuales consideraron que sería bueno utilizar simulación para la administración de proyectos múltiples son las siguientes:

- Los proyectos múltiples son intrínsecamente complejos por su naturaleza debido a las interdependencias de recursos, tamaño, prioridades y niveles de progreso.

- La simulación es una herramienta útil para la comprensión cuidadosa de fenómenos complejos y para conducir estudios que no se pueden realizar fácilmente con casos reales.

En su estudio también consideraron las siguientes características fundamentales para un ambiente de proyectos múltiples:

- Los proyectos múltiples son interdependientes debido al uso de un recurso común.
- Algunos métodos deben ser empleados para priorizar el uso de recursos entre proyectos múltiples.
- Hay intercambios complejos entre la utilización de los recursos y la terminación de los proyectos individuales a tiempo.
- Debe existir un mecanismo de control ya sea de organización o táctico para reducir la variación entre las fechas planeadas y reales de terminación del proyecto.

Al utilizar simulaciones, el administrador del proyecto puede explorar la táctica de administración de los proyectos múltiples en diversos modos. El modelo de simulación de proyectos múltiples puede proveer los siguientes beneficios a los administradores de proyecto:

- Los administradores pueden valorar los efectos de recursos de contenido de varios proyectos.
- Los administradores pueden entender las interdependencias de los proyectos múltiples, la relación causal que influye varios proyectos, y las consecuencias en la fecha de terminación de cada proyecto.

- Los administradores deben considerar diferentes escenarios de posibilidades para evaluar los efectos de diferentes políticas en la programación de calendario y la utilización de recursos.
- Los nuevos administradores pueden utilizar el modelo como un simulador para entrenarse a sí mismos con diferentes condiciones, sin el costo de aprender con un proyecto real.

En la Figura 1.2, se muestra el modelo seguido por Miller y Lee para la administración de proyectos múltiples utilizando el método de la cadena crítica y el modelo de sistema dinámico, Miller y Lee consideraron contar con cuatro componentes para su modelo de simulación que son el modelo del escenario, modelo de sistema dinámico, modelo de red Multi-Proyecto, y un controlador, estos componentes se explican a detalle en el capítulo 2.

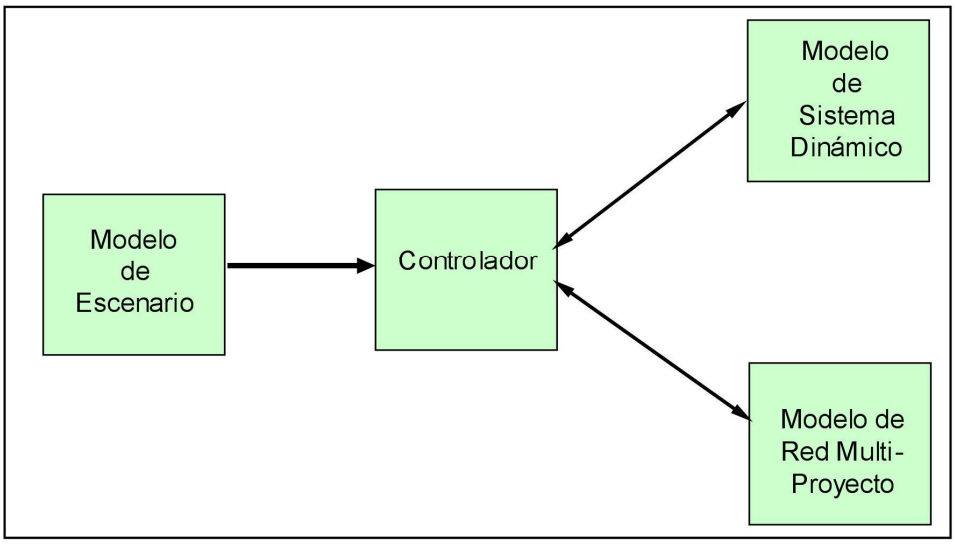

Figura 1.2. Modelo de simulación para la administración de proyectos múltiples de Miller y Lee.

Cho y Eppinger (2005), presentan un modelo y análisis de técnicas para la administración de diseño de proyectos. Dicho modelo calcula la distribución de la probabilidad a través del tiempo de manera estocástica. Se utiliza un análisis basado en simulación para explicar aspectos realistas del comportamiento del proceso de diseño que no eran posibles en modelos analíticos anteriores.

1.2 Planteamiento del Problema

En la actualidad existen métodos para dar seguimiento a las actividades de un proyecto de software tales como ruta crítica, cadena crítica, diagramas, PERT etc. También existen metodologías para gestionar los riesgos que se presentan en un proyecto de software incluidas en procesos, como RUP y MSF. Cada uno de estos métodos ofrece ciertas ventajas, que el administrador de proyectos analizará para decidir con cual de estos métodos es conveniente administrar su proyecto de software.

Existen propuestas para crear un software simulador para dar seguimiento a las actividades del proyecto y administrar los riesgos bajo arquitecturas que contemplan un modelo específico como la arquitectura del estudio de Padberg y la de Miller y Lee. Estas arquitecturas no contienen componentes independientes del modelo de simulación, son arquitecturas rígidas y están limitadas.

En este tipo de arquitecturas se propone realizar una simulación, bajo un método de seguimiento de actividades determinado, y el administrador de proyectos tendrá que estar familiarizado con dicha metodología para poder dar seguimiento a su proyecto. En la actualidad no existe una estrategia o método dominante para el seguimiento de actividades y para realizar las gestiones de riesgos de un proyecto.

1.3 Justificación

Todo software debe contar con ciertas características que le permitan ser competitivo, por ejemplo: Flexibilidad y Escalabilidad. El diseño de una buena arquitectura de software permite tener software flexible y reutilizable. De ésta manera, cualquier componente puede ser sustituido o modificado sin que éste afecte al resto del programa [Pressman, 2002].

La arquitectura global de un sistema es importante para implementar, probar y dar mantenimiento al sistema de una forma más sencilla. La calidad de la arquitectura puede hacer o deshacer al software. La estructura arquitectónica debe reflejar los principios de un buen diseño [Lawrence, 2002].

Una arquitectura de software es la estructura base de un software, la cual contempla componentes y propiedades, así como su interacción entre ellos. Una arquitectura flexible permite diseñar software que sea reutilizable, flexible y escalable. Existen diversos métodos para administrar los tiempos de

un proyecto, así que es necesario contar con una arquitectura flexible, con interfaces bien definidas que permitan sustituir componentes de la arquitectura sin que el resto de los componentes ni el funcionamiento global del software se vea afectado.

No se han encontrado trabajos centrados en la arquitectura, que tomen en cuenta la diversidad de técnicas y métodos involucrados en el seguimiento temporal de los proyectos de software.

1.4 Objetivo General

El objetivo general de este trabajo es definir una arquitectura flexible, para un simulador de seguimiento de calendario temporal de proyectos de software con componentes intercambiables, que contemple análisis de riesgos y el seguimiento de actividades.

Objetivos Particulares

1. Realizar un estudio de los diversos aspectos involucrados en el seguimiento temporal de los proyectos de software.
2. Identificar el impacto del manejo de riesgos en el calendario de los proyectos de software y su repercusión en la arquitectura.

1.5 Estructura de la tesis

Se realizó un estudio referente a la administración de proyectos de software, métodos para la administración de actividades, gestión de riesgos en los proyectos, modelos de simulación, y arquitecturas propuestas para la creación de un simulador de proyectos de software. El presente trabajo se fundamenta en el hecho de que existen diferentes métodos para el seguimiento temporal de proyectos, para el análisis de los factores de riesgos, y modelos de simulación de proyectos. El desarrollo de este proyecto está distribuido de la siguiente manera:

- **Capítulo II.** Este capítulo contiene bibliografía, con especial énfasis en :
    - Administración de proyectos
    - Procesos de desarrollo de software
    - Métodos para la administración de proyectos
    - Métodos para gestión de riesgos
    - Arquitectura del software
    - Simulación y modelado de escenarios
    - Previos estudios y publicaciones relacionadas con este trabajo.

- **Capítulo III.** Este capítulo describe la metodología seguida para el desarrollo de la investigación. Se divide en tres aspectos, la investigación documental, investigación de campo y generación de una arquitectura.

- **Capítulo IV.** Este capítulo describe el desarrollo de la arquitectura propuesta para el simulador de seguimiento temporal de actividades. Se presentan los Escenarios y Casos de uso, también se muestran los diagramas de casos de uso, diagrama de secuencia, diagramas de componentes y diagrama de clases.

- **Capítulo V.** En este capítulo se muestran las conclusiones de la investigación y las recomendaciones para trabajo futuro.

CAPÍTULO II. MARCO TEÓRICO

2.1 Administración de proyectos

La administración de proyectos es la aplicación de conocimientos, habilidades, técnicas y herramientas a las actividades de un proyecto, con el fin de satisfacer, cumplir y superar las necesidades y expectativas de los involucrados [Yamal, 2002].

En la Tabla 2.1, se muestran las principales actividades para la administración de proyectos que son: planificación, organización, dotación de personal, dirección y control.

ACTIVIDAD	DEFINICIÓN
Planificación	Predeterminar un curso de acción para lograr los objetivos de la organización
Organización	Arreglar y relacionar el trabajo para lograr los objetivos y para delegar responsabilidad y autoridad para lograr esos objetivos
Dotación de personal	Seleccionar y capacitar al personal para los puestos en la organización
Dirección	Crear una atmósfera que ayudará y motivará a la gente para lograr los resultados finales deseados
Control	Medir y corregir el rendimiento de las actividades que se dirigen hacia los objetivos, de acuerdo a lo planificado

Tabla 2.1. Principales actividades en la administración de proyectos.

Hoy en día, se cuenta con innumerables métodos para la administración de proyectos, pero es desde hace poco que se han ido analizando por parte de

los investigadores operacionales, los problemas gerenciales asociados. La estructura desagregada del trabajo, los paquetes de trabajo, los diagramas de red, los diagramas de Gantt, y las redes PERT/CPM constituyen recursos necesarios para completar la actividad en el menor tiempo posible y con el mínimo de fallas.

Estos métodos ayudan a identificar los instantes del proyecto en que estas restricciones causarán problemas, y de acuerdo a la flexibilidad permitida por los tiempos de holgura de las actividades no críticas, permiten que el administrador manipule ciertas actividades para resolver estos problemas. Es importante que los nuevos administradores conozcan todos estos aspectos, debido a que son factores de éxito importantes para las organizaciones y garantizan que se logren los objetivos del proyecto en el tiempo previsto y con el presupuesto asignado [Noori y Hamid, 1997].

2.2 Proceso de desarrollo de software

Un proceso define quién está haciendo qué, cuándo, y cómo alcanzar un determinado objetivo [Booch, *et al* 2000]. En la ingeniería de software el objetivo es construir un producto de software o mejorar uno existente. Un proceso efectivo proporciona normas para el desarrollo eficiente de software de calidad; captura y presenta las mejores prácticas que el estado actual de la tecnología permite.

En consecuencia, reduce el riesgo y hace el proyecto más predecible. Para construir software de calidad se necesita un proceso que sirva como guía para todos los participantes dentro del proyecto (usuarios, desarrolladores, directivos) para que puedan comprender el rol y la importancia que tienen dentro del desarrollo del sistema.

2.3 La administración de proyectos en el contexto de los procesos de desarrollo de software

Existen procesos de software que contemplan de manera explícita apartados especiales para la administración de los proyectos. Dos ejemplos de estos son RUP y MSF.

2.3.1 Rational Unified Process, (*Proceso Unificado de Desarrollo-RUP)*

El RUP junto con el lenguaje unificado de modelado (UML), constituyen una metodología estándar utilizada para el análisis, implementación y documentación de sistemas orientados a objetos, está basada en seis principios clave [Booch, *et al* 2000]:

1. **Adaptar el proceso.** El proceso deberá adaptarse a las características propias del proyecto u organización, tamaño del mismo, así como las restricciones que lo regulen.
2. **Balancear prioridades.** Los requerimientos de los diversos clientes pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos.
3. **Elaboración entre equipos.** El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados, etc.
4. **Demostrar valor iterativamente.** Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión del cliente, la estabilidad y calidad del producto.
5. **Elevar el nivel de abstracción.** Es conveniente realizar el software empleando o creando herramientas reutilizables, que no sirvan sólo para el proyecto actual sino también para proyectos futuros.
6. **Enfocarse en la calidad.** El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción.

2.3.1.1. Ciclo de Vida del RUP

El ciclo de vida del RUP es una implementación del desarrollo en espiral. El RUP divide el proceso de desarrollo en cuatro fases: Inicio, Elaboración, Construcción y Transición, teniendo un producto al final de cada iteración. Durante las fases del proyecto se desarrollan las actividades que se muestran en la tabla 2.2 [Booch, *et. al.* 2004].

Fase	Actividades a Realizar
Inicio	Modelado de negocio Captura de requerimientos
Elaboración	Refinamiento de requerimientos Análisis y diseño del sistema
Construcción	Refinamiento del diseño Implementación
Transición	Pruebas Despliegue Administración de configuración y cambios

Tabla 2.2. Ciclo de vida del proyecto.

Estas fases a su vez se dividen en iteraciones, cada una de las cuales produce una pieza de software demostrable. A través de las fases se desarrollan en paralelo nueve disciplinas, como se muestra en la Figura 2.1.

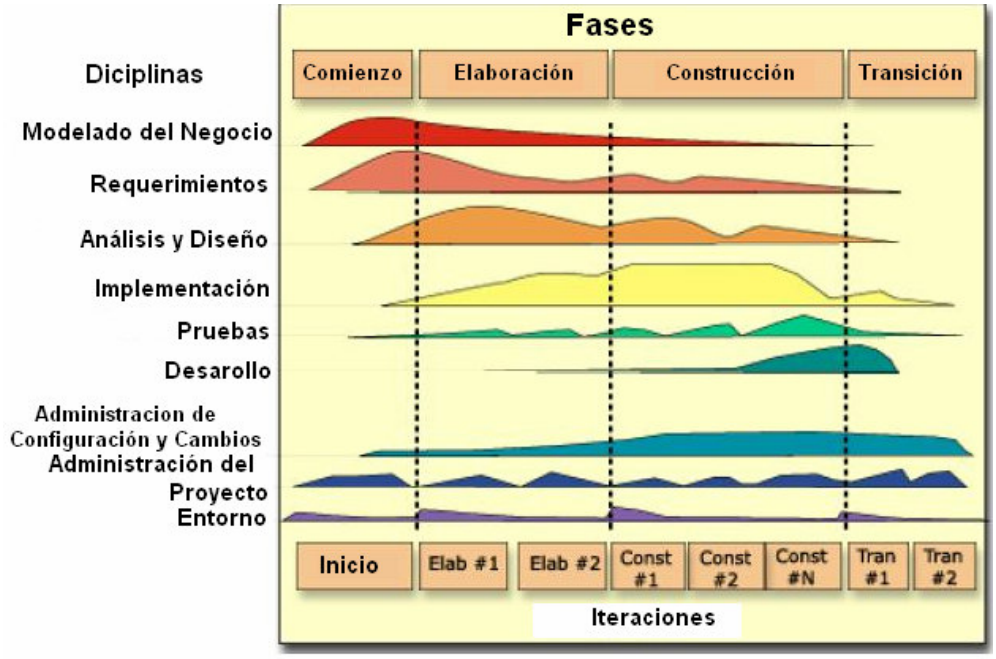

Figura 2.1. Fases y disciplinas del ciclo de vida del proyecto.

2.3.1.2 Administración del proyecto

La administración del proyecto se enfoca en la planeación del proyecto de software. Se genera un plan para el desarrollo del software, un plan del proyecto y un plan de ejecución del proyecto, para la fase de inicio, elaboración y construcción como se explica a continuación:

- ***Plan de desarrollo del software.*** El objetivo de este plan es proporcionar la información necesaria para controlar el proyecto. En él se describe el enfoque de desarrollo del software.

- ***Plan del Proyecto.*** Se crea un plan para el proyecto durante la fase de inicio, elaboración y construcción para todo el proyecto.

- ***Plan de ejecución del proyecto.*** Se lleva a cabo una bitácora diaria de las actividades que se realizan en el proyecto la fecha, actividad desarrollada y el tiempo de duración.

2.3.2 Microsoft Solutions Framework *(Marco de Trabajo de Soluciones - MSF)*

MSF, es un conjunto de modelos, disciplinas, conceptos, lineamientos y mejores prácticas probadas, las cuales fueron elaboradas por Microsoft. MSF reconoce la naturaleza *caórdica* (una combinación de caos y orden) de los proyectos de tecnología. Toma como punto de partida el

supuesto de que debe esperarse un cambio continuo y de que es imposible aislar un proyecto [Microsoft TechNet, 2006].

En la Figura 2.2, se pueden observar las cuatro fases que debe cumplir un proyecto de software según la metodología de MSF: Visualizar, Planificar, Desarrollar y Estabilizar que se explican a continuación.

Figura 2.2. Fases del MSF.

Visualizar. En ésta fase se elaboran los siguientes documentos:

- Elaboración y aprobación del documento de alcance y estrategia definitivo.
- Formación del equipo de trabajo y distribución de competencias y responsabilidades.
- Elaboración del plan de trabajo.
- Elaboración de la matriz de riesgos y plan de contingencia.

Planificar. *S*e realiza la planificación del proyecto generando los documentos siguientes:

- Documento de planificación y diseño de arquitectura
- Documento de plan de laboratorio - prueba de concepto

Desarrollar. Se llevan a cabo en esta fase los planes diseñados en la anterior, principalmente el de despliegue y el de formación. Los principales trabajos e hitos a conseguir son en este caso:

- Implantación de la plataforma.
- Formación a los usuarios y administradores.
- Registro de mejoras y sugerencias.
- Entrega de documentos definitivos.
- Entrega del proyecto.

Estabilizar. La solución implantada pasa a un entorno real de explotación, restringido en número de usuarios y en condiciones tales que se pueda llevar un control efectivo de la situación. Los hitos y objetivos fundamentales de esta fase son:

- Selección del entorno de prueba piloto.
- Gestión de Incidencias.
- Revisión de la documentación final de Arquitectura.
- Elaboración de la documentación de Formación y Operaciones.
- Elaboración del Plan de Despliegue.
- Elaboración del Plan de Formación.

MSF ha diseñado tanto su modelo de equipo como su modelo de proceso, para anticiparse al cambio y controlarlo. Emplea el principio fundamental de

permanecer ágil ante un cambio y utiliza la experiencia como oportunidad de aprendizaje. Los ocho principios fundamentales de MSF son los siguientes:

1. Alimentar comunicaciones abiertas.
2. Trabajar hacia una visión compartida.
3. Otorgar poder a los miembros del equipo.
4. Establecer responsabilidad clara y compartida.
5. Concentrarse en la entrega de valor de negocios.
6. Esperar el cambio.
7. Invertir en calidad.
8. Aprender de todas las experiencias.

2.3.2.1 Características de la administración de proyectos MSF

Microsoft Solutions Framework (MSF) cuenta con un enfoque de equipo distribuido para llevar a cabo la administración de proyectos. Algunos proyectos grandes o complejos necesitan un administrador de proyectos o un equipo de administración de proyectos especializado. MSF da mucha importancia a la disciplina y las competencias asociadas a la administración de proyectos y categoriza las siguientes áreas de responsabilidades, técnicas y actividades en la administración de proyectos:

- Administrar y realizar un seguimiento de los cambios.
- Administración de la programación
- Administración de costos

- Administración de los recursos humanos
- Administración del riesgo
- Administración de la calidad

2.4 Métodos para la administración de proyectos

Existen diferentes tipos de métodos para la administración de proyectos. Estos son solo algunos ejemplos: PERT, CPM, Ruta Crítica y Cadena Crítica. Todos los métodos tienen como objetivo el administrar las actividades del proyecto, siguiendo patrones y programaciones específicas.

2.4.1 Program Evaluation and Review Tecnique, (*Técnica de Revisión y Evaluación de Programas - PERT).*

PERT fue desarrollado por científicos de la Oficina Naval de Proyectos Especiales que propusieron este modelo para controlar los tiempos de ejecución de las diversas actividades integrantes de los proyectos espaciales, y fue utilizado originalmente por el proyecto Polaris. Es un instrumento diseñado especialmente para la dirección, permitiéndole planificar, programar y controlar los recursos de que dispone con el fin de obtener los resultados deseados [Rooco, 1967].

Este método cuenta con las siguientes actividades para administrar un proyecto:

- Definir el proyecto y todas las actividades o tareas importantes.
- Determinar las relaciones entre las actividades.
- Decidir qué actividades deben preceder a otras y cuáles deben seguir a otras.
- Esbozar una red que conecte todas las actividades.
- Asignar tiempo y/o costos estimados a cada actividad.
- Calcular el tiempo requerido para completar las actividades en cada trayecto de la red.
- Emplear la red para que sirva de soporte al plan, el programa, el monitoreo y el control del proyecto.

2.4.2 Critical Path Method*, (Método de Camino Crítico - CPM).*

CPM es un proceso administrativo de planeación, programación y control de todas y cada una de las actividades del proyecto. Fue desarrollado en 1957, buscando el control y la optimización de los costos de operación mediante la planeación adecuada de las actividades componentes del proyecto.

Este modelo cuenta con un enfoque completo y probado en el entorno de la Administración de Proyectos, que no se basa en la administración de un proyecto en función de teorías de estimaciones “seguras” de tareas. Este

método es muy similar al PERT. La única diferencia con CPM es relacionada con la duración estimada de cada actividad: CPM emplea un estimado de tiempo de un sólo punto de duración de la actividad, mientras que PERT utiliza tres (muy probable, probable, lo menos probable) [Montaño, 2003].

2.4.3 Ruta Crítica

La Ruta Crítica surge de la fusión de dos métodos PERT (maneja los tiempos inciertos de las actividades del proyecto) y CPM (maneja tiempos conocidos de las actividades del proyecto). Las actividades que no están en la ruta crítica tienen una cierta cantidad de holgura; es decir, pueden empezar más tarde y permiten que el proyecto como un todo se mantenga conforme a lo programado. El PERT/CPM identifica estas actividades y la cantidad de tiempo disponible para retardos. Actualmente se ha tomado lo mejor de ambos métodos y se han vuelto uno solo, conocido como Método de la Ruta Crítica [Anderson, et al, 1999].

El PERT/CPM fue diseñado para proporcionar diversos elementos útiles de información para los administradores de proyectos. Este método expone la ruta crítica de un proyecto; esto es, las actividades que limitan la duración de un proyecto. En otras palabras, para lograr que el proyecto se realice pronto, las actividades de la ruta crítica deberán realizarse pronto. Por otra parte, si una actividad de la ruta crítica se retrasa, el proyecto como un todo se retrasará en la misma cantidad.

2.4.4 Cadena Crítica

Entre los métodos orientados al seguimiento del proceso de los proyectos existe una denominada Cadena Crítica. La Cadena Crítica está enfocada al manejo de la incertidumbre inherente a todo proyecto. Se define como una cadena crítica a la secuencia de eventos dependientes que evitan que el proyecto se complete en un intervalo más corto de tiempo. Donde un evento dependiente es aquel que utiliza como insumo tareas que otro evento produce o utiliza recursos que otro evento esta ocupando [Goldratt, 2000].

El método de Cadena Crítica inicia con la construcción de una red de actividades compuesta por varias cadenas de actividades dependientes a la cadena crítica en algún punto del calendario. Una característica del método es que evita agregar tiempo de protección a cada actividad, se inicia con una estimación de la duración de cada actividad sin incluir protección. Al final de cada cadena se agrega la protección. El control del proyecto se hace monitoreando el consumo de ésta protección. Con ese mecanismo se evita que el equipo de desarrollo tenga que reaccionar a falsas alarmas debido a pequeñas variaciones en el proyecto.

Las reglas para llevar a cabo el método de cadena crítica son las siguientes:

- No se establecen fechas fijas de inicio y fin de cada actividad, solo se define la fecha de inicio de cada cadena de actividades y la fecha final de entrega del proyecto.

- Las personas participantes estarán concentradas en una sola actividad a la vez, es decir, está prohibido participar en varias actividades simultáneamente.

- Cada persona deberá iniciar la actividad que le corresponda cuando la actividad de la cual depende sea completada.

- El administrador se enfocará en la supervisión del consumo de amortiguadores de tiempo, tomando las acciones pertinentes cuando estos consumos sobrepasen limites establecidos.

Los pasos para la construcción del método de cadena crítica se enumeran a continuación:

- Desarrollar la red de actividades para cada proyecto
- Dar prioridad a recursos tomando en cuenta la disponibilidad entre proyectos
- Identificar la cadena crítica por proyecto
- Ubicar los amortiguadores en cada proyecto

El cronograma de un proyecto puede ser diseñado para proteger la fecha de terminación a través de tomar amortiguadores que estaban distribuidos en todas las tareas y concentrar esa contingencia o seguridad en el lugar donde más hace falta, al final de la cadena crítica y donde otras cadenas alimenten la cadena crítica.

En la Figura 2.3, se observa que es agregado un tiempo de protección por actividad en el proyecto, utilizando un método convencional para el seguimiento de actividades, y a su vez el administrador de proyectos, agrega un tiempo de protección a todo el proyecto.

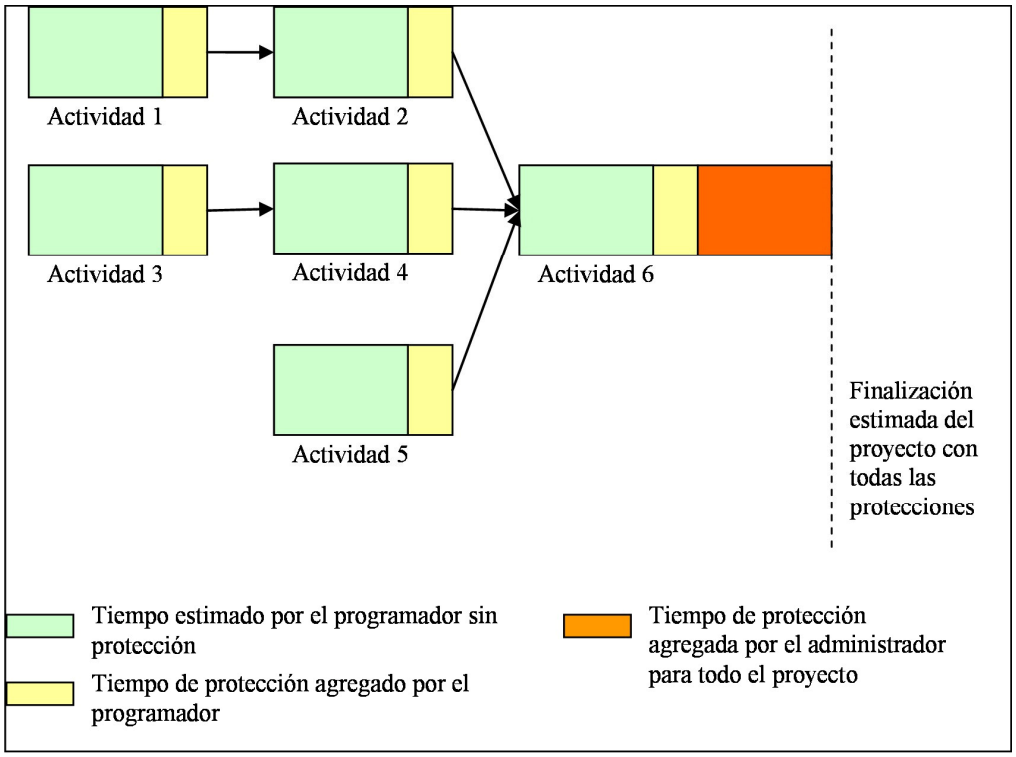

Figura 2.3. Asignación del tiempo de protección por tarea.

En la Figura 2.4, se muestra como es distribuido el tiempo de protección en el proyecto utilizando el método de cadena crítica. Se puede observar que ya no existe un tiempo de protección por actividad, como en el método tradicional, solo al final del proyecto y al final de las cadenas alimentadoras.

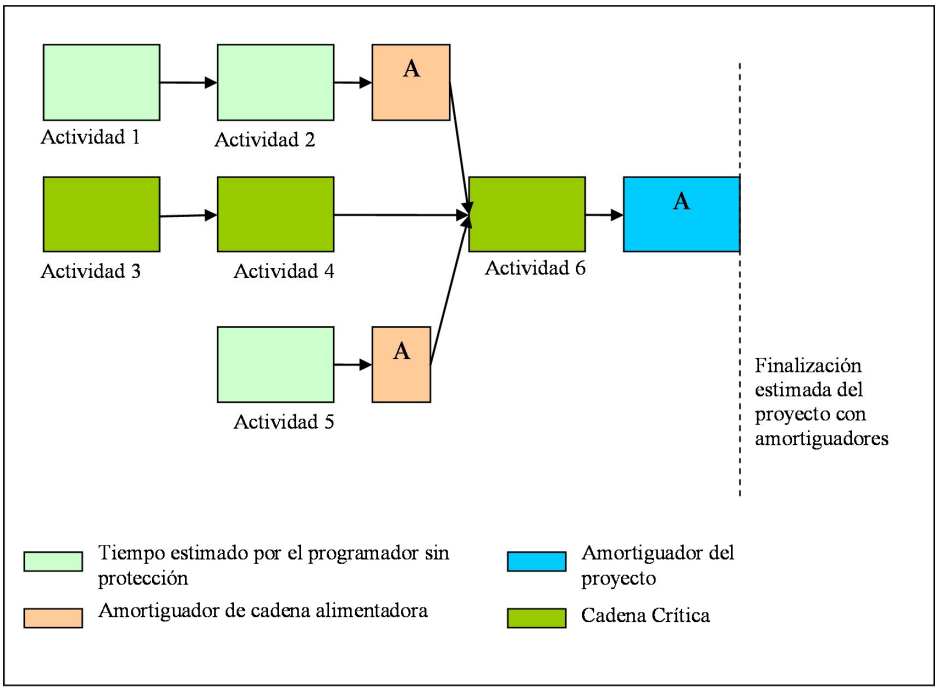

Figura 2.4. Asignación de tiempos de protección por proyecto utilizando cadena crítica.

2.5 Gestión de riesgos

Los Riesgos son los eventos desagradables no deseados, que ocurren durante el desarrollo del software [Lawrence, 2002]. La mayoría de los administradores de proyectos de software utilizan etapas para asegurar que sus proyectos se realicen en el tiempo dentro de las restricciones de esfuerzo y costo. Sin embargo, la gestión de proyectos involucra más que el seguimiento del esfuerzo y del cronograma. Los administradores deben determinar si pueden presentarse eventos no deseados durante el desarrollo o el mantenimiento y hacer planes para evitar dichos eventos, o si éstos son inevitables, minimizar sus consecuencias negativas. Un riesgo es un evento no deseado que tiene consecuencias negativas. Los administradores de proyectos

deben ocuparse de la gestión del riesgo para comprender y controlar los riesgos en sus proyectos.

2.5.1 Definiciones de Riesgo

- Un riesgo es un evento no deseado que tiene consecuencias negativas. Un riesgo son muchos eventos que ocurren durante el desarrollo del software por ejemplo:
  1. Una pérdida asociada a un evento. El evento crea una situación donde al proyecto le suceden algunas cosas negativas; pérdidas de tiempo, calidad, dinero, control, comprensión etc.
  2. La probabilidad de que el evento pueda ocurrir.
  3. El grado en que se puede cambiar el resultado. Para cada riesgo, se debe determinar que se puede hacer para minimizar o evitar el impacto del evento [Lawrence, 2002].

- El riesgo afecta a los futuros acontecimientos, el riesgo implica cambios que pueden venir por cambios de opinión, de acciones, lugares etc. El riesgo implica elección, y la incertidumbre que entraña la elección. Por lo tanto, el riesgo es una de las pocas cosas inevitables de la vida [Pressman, 2002].

2.5.2 Elementos de la Gestión de Riesgos

La función de la gestión de riesgos del software es identificar, estudiar y eliminar las fuentes de riesgo antes de que empiecen a amenazar la finalización satisfactoria de un proyecto de software. Puede controlar los riesgos a varios niveles: control de crisis, arreglar cada error, mitigación de riesgos y prevención. La Tabla 2.3, muestra la descripción de cada uno de estos niveles. Generalmente, la gestión de riesgos se divide en estimación de riesgos y control de riesgos [Ropponen, Lyytinen, 2000].

NIVEL	DESCRIPCIÓN
Control de crisis	Controlar los riesgos sólo cuando se han convertido en problemas
Arreglar cada error	Detectar y reaccionar rápidamente ante cualquier riesgo, pero sólo después de que se haya producido
Mitigación de riesgos	Planificar con antelación el tiempo que necesitaría para cubrir riesgos en el caso de que ocurran, pero no intentar eliminarlos inicialmente
Prevención	Crear y llevar a cabo un plan como parte del proyecto software para identificar riesgos y evitar que se conviertan en problemas
Eliminación de las causas principales	Identificar y eliminar los factores que puedan hacer posible la presencia de algún tipo de riesgo

Tabla 2.3. Niveles de la gestión de riesgo.

2.5.3 Actividades de la Gestión de Riesgos

El control de riesgo involucra un conjunto de acciones tomadas para reducir o eliminar un riesgo. La gestión de riesgos involucra varias etapas importantes, entre las que se encuentran la identificación del riesgo, análisis de riesgos, priorización del riesgo, reducción del riesgo, planificación de la gestión del riesgo, y resolución del riesgo [Lawrence, 2002]. En la Figura 2.5, se muestran las dos categorías de la gestión de riesgos (Estimación de riesgos y Control de riesgos); y éstas, se dividen en seis subcategorías, que son:

identificación, análisis, prioridad, planificación resolución y monitoreo; como se detalla más adelante.

Figura 2.5. Gestión de riesgos.

Estimación de Riesgos

En la estimación de riesgos se identifica el riesgo y se realiza un análisis para determinar si es factible que el riesgo pueda afectar la fecha de terminación del proyecto y cuáles son los riesgos que tendrían mayor impacto si se presentaran. Los pasos a seguir para la estimación del riesgo son los siguientes:

a) ***Identificación del riesgo.*** Es el primer paso en la gestión de riesgos. Se identifican los factores que introducen un riesgo en la planificación y se genera una lista de riesgos capaces de poner en peligro la planificación del proyecto.

b) ***Análisis de riesgos.*** Una vez que se hayan identificado los riesgos de planificación del proyecto, el siguiente paso es analizar cada riesgo para determinar su impacto. Se utiliza el análisis de riesgos para poder elegir entre varias alternativas de desarrollo o para gestionar los riesgos asociados con una alternativa que ya se haya elegido. El análisis de riesgos mide la probabilidad y el impacto de cada riesgo y los niveles de riesgos de los métodos alternativos.

c) ***Priorización de riesgo.*** Después de que se haya creado la lista de los riesgos de la planificación, el siguiente paso es priorizar los riesgos de forma que se conozca donde centrar el esfuerzo para la gestión de riesgos. Los proyectos gastan generalmente el 80 por ciento del presupuesto en arreglar el 20 por ciento de los problemas, por lo que es útil poder centrarse en este 20 por ciento más importante.

Control de Riesgos

Una vez que se identifique los riesgos del proyecto, analizando las probabilidades y magnitudes y se hayan priorizado, se está preparado para controlarlos. Los tres aspectos en el control del riesgo son los siguientes:

a) ***Planificación de la gestión del riesgo.*** El objetivo de la planificación es desarrollar un plan que controle cada uno de los riesgos de prioridad alta identificados en las actividades anteriores, describiendo quién, qué, cuándo, y cómo se gestiona cada uno de los riesgos.

b) ***Resolución de riesgos.*** Es la ejecución del plan para resolver cada uno de los riesgos significativos. Depende mucho del riesgo específico. Métodos para tratar a los riesgos:

- Evitar los riesgos
- Trasladar el riesgo de una parte del sistema a otra
- Conseguir información sobre el riesgo
- Eliminar el origen del riesgo
- Asumir el riesgo
- Comunicar el riesgo

- Controlar el riesgo
- Recordar el riesgo

c) ***Monitoreo de riesgos.*** Es la actividad del progreso de monitoreo dirigido a la resolución de cada elemento del riesgo. El monitoreo del riesgo también puede incluir la continuación de la actividad de la identificación de nuevos riesgos y volver a considerarlos en el proceso de la gestión de riesgos.

La Figura 2.6, muestra como se gestionan los riesgos en el desarrollo del software.

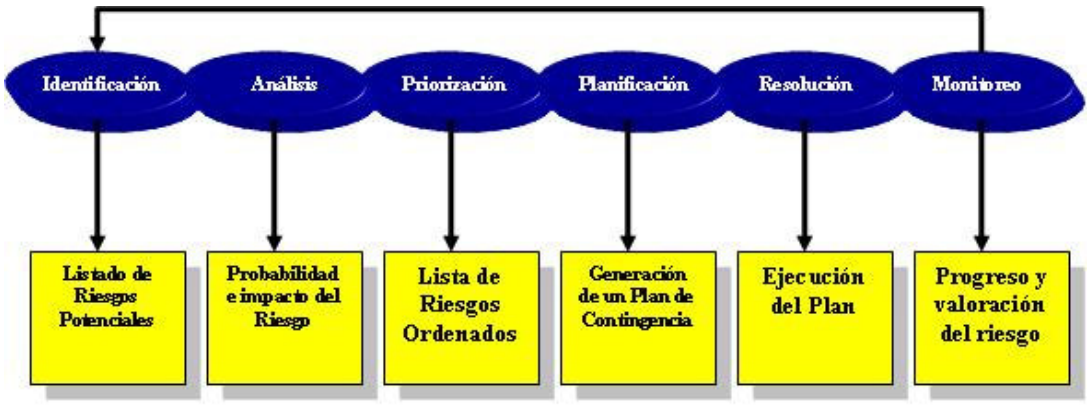

Figura 2.6. Gestión de Riesgos en el desarrollo del software.

Algunos de los riesgos más frecuentes en el desarrollo del software son: la falta de personal, programación de tiempos poco realista, inadecuado desarrollo de funciones del software, continuos cambios en los requerimientos, etc.

En la Tabla 2.4, se muestran los diez riesgos más frecuentes en el desarrollo de software y su descripción según la clasificación de [Boehm, 1991].

	Riesgo	Descripción
1	Déficit de personal	Carencia de personal calificado
2	Presupuestos y programación poco realista	Tiempo de desarrollo y presupuesto incorrecto (muy bajos)
3	Funciones del software desarrolladas incorrectamente	Desarrollo de funciones del software que no son necesarias o están mal especificadas
4	Desarrollar incorrectamente las interfaces de usuario	Interfaz inadecuada o difícil de usar
5	Bandeja de oro	Adición de características innecesarias para el software por orgullo profesional o por demandas del usuario
6	Continuos cambios en los requerimientos	Cambios no controlados de las funciones o rasgos del sistema
7	Deficiencias en los componentes desarrollados externamente	Mala calidad de los componentes del sistemas que se han desarrollado por terceros
8	Deficiencias en tareas externamente realizadas	Mala calidad o realización impredecible de las tareas que se realizan fuera de la organización
9	Déficit de programación en tiempo real	Realización pobre de los resultados del sistema
10	Abusos de las capacidades de la informática	Inhabilidad de implementar el sistema por falta de soluciones técnicas y cómputo poderoso

Tabla 2.4. Riesgos más frecuentes en proyectos de software según la clasificación de Boehm.

En la creación de escenarios de un proyecto es necesario contemplar el factor de incertidumbre asociado a sucesos inesperados. Ropponen y Lyytinen (2000) tomaron los diez riesgos más importantes definidos por Boehm, como se muestra en la Tabla 2.4, y los reclasificaron de la siguiente forma:

1. **Riesgos de Programación y tiempos.** Son riesgos relacionados con una mala programación en el tiempo estimado de desarrollo del proyecto software.
2. **Riesgos en la funcionalidad del sistema.** El sistema puede contar con funciones que no fueron especificadas en los requerimientos o que no son necesarias.
3. **Riesgos de Subcontratación.** Están relacionados con la calidad de los componentes y tareas realizadas por terceros. Experiencia en proyectos

grandes, experiencia en administración de proyectos y el tamaño de la organización son los factores que influyen en este riesgo.

4. **Riesgos de la administración de requerimientos.** Están relacionados con la capacidad del administrador de proyectos para manejar el cambio de los requisitos, para evitar una mala planeación y no agregar características innecesarias o que no agregan valor al producto.
5. **Riesgos del uso y desempeño de los recursos.** La experiencia del administrador de proyectos para estimar los recursos necesarios con que debe de contar para la realización del proyecto, es la variable con más peso dentro del análisis de riesgos y un factor que determina el buen uso de los recursos.
6. **Riesgos de administración de personal.** En las últimas etapas de un proyecto es importante no poner bajo tensión al personal para evitar los riesgos por falta de personal, tiempo consumido constante, falta de experiencia, falta de personal calificado, etc.

Ropponen y Lyytinen identificaron que los factores del medio ambiente y prácticas en la administración influyen directamente en el análisis del riesgo, como se muestra en la Figura 2.7.

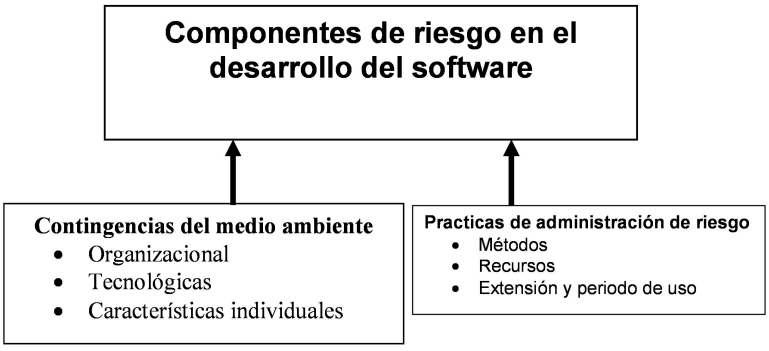

Figura 2.7. Componentes de riesgo en el desarrollo de software.

2.6 Arquitectura del Software

Se puede pensar que la arquitectura de un sistema es la visión común en la que todos los empleados, desarrolladores, y usuarios, deben estar de acuerdo. La arquitectura da una clara perspectiva del sistema completo, necesaria para controlar el desarrollo del sistema [Booch, *et al* 2000].

En la actualidad, existen diversos estudios que proponen arquitecturas para la creación de sistemas de cómputo y sistemas complejos. Otros tantos se basan en arquitecturas ya probadas y que han funcionado correctamente de acuerdo a los requerimientos del sistema como son: Cliente – Servidor, Arquitectura de Base de Datos, Pipas, Broker, etc. Algunos de los beneficios que se ganan con el surgimiento de la arquitectura del software es una mayor disciplina de:

- Soporte para satisfacer los requerimientos
- Técnica base del diseño y base directiva de estimación de costos y administración de procesos.
- Base efectiva para la reutilización.
- Base para la dependencia y consistencia del análisis [Garlan y Shaw, 1994].

La arquitectura de software abarca decisiones importantes sobre:

- La organización del sistema de software.

- Los elementos estructurales que compondrán el sistema y sus interfaces, junto con sus comportamientos, tal y como se especifican en las colaboraciones entre estos elementos.
- La composición de los elementos estructurales y del comportamiento en subsistemas progresivamente más grandes.
- El estilo de la arquitectura que guía esta organización: los elementos y sus interfaces, sus colaboraciones y su composición.

Sin embargo, la arquitectura de software está afectada no sólo por la estructura y el comportamiento, sino también por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de compresión, las restricciones, compromisos económicos y tecnológicos, y la estética [Booch, *et al* 2000].

La arquitectura es una estructura organizativa de un sistema que incluye su descomposición en partes, conectividad, mecanismos de interacción y principios de guía que proporcionan información sobre el diseño del mismo [Booch, *et. al.* 1999].

La arquitectura es el conjunto de decisiones significativas sobre la organización de un sistema de software que incluye:

- La selección de elementos estructurales y las interfaces mediante las que se conectan: la organización a gran escala, los elementos estructurales y la topología de su conexión.
- Su comportamiento en las colaboraciones entre dichos elementos.

- Los mecanismos importantes de que se dispone en el sistema y el estilo arquitectónico que guía su organización

Las decisiones arquitectónicas sobre la descomposición de un sistema en partes se pueden capturar utilizando modelos, subsistemas, paquetes, y componentes. Las dependencias entre estos elementos son indicadores clave de la flexibilidad de la arquitectura y de la dificultad de modificar el sistema a futuro. Los mecanismos que proporciona una arquitectura para construir sobre ella deben de ser capturados mediante patrones y colaboraciones [Booch, *et al.* 1999].

La arquitectura global de un sistema es importante no solo porque es fácil implementarla y probarla sino también en cuanto a la velocidad y efectividad del mantenimiento y el cambio. La calidad de la arquitectura puede hacer o deshacer un sistema. Entre más independientes son las unidades de la arquitectura, más modular se vuelve la arquitectura y más fácilmente se pueden diseñar y desarrollar las piezas por separado. Existen al menos cinco formas para dividir un sistema en unidades [Lawrence, 2002]:

1. Descomposición modular, basada en la asignación de funciones a los módulos.

2. Descomposición orientada por datos, basada en la estructura externa de los datos.

3. Descomposición orientada por eventos, basada en eventos que el sistema debe tratar.

4. Diseño de afuera hacia dentro, basada en las entradas del usuario al sistema.

5. Diseño orientado a objetos, basado en la identificación de clases de objetos y sus interrelaciones.

2.6.1 ¿Por qué es necesaria la Arquitectura?

Un sistema de software grande y complejo requiere una arquitectura para que los desarrolladores puedan progresar hasta tener una visión común del proyecto a desarrollar [Booch, *et al* 2000].

Un sistema de software es difícil de abarcar visualmente porque no existe en un mundo de tres dimensiones, es a menudo único y sin precedente en determinados aspectos. Suele utilizar tecnología poco probada o una mezcla de tecnologías nuevas, tampoco es raro que el sistema lleve a sus últimos límites la tecnología existente. Además, debe ser construido para acomodar gran cantidad de clases que sufrirán cambios a futuro. A medida que los sistemas se hacen más complejos, existe con frecuencia un sistema que ya realiza algunas de las funciones del sistema propuesto. Si no se tiene claro que debe de hacer el sistema, se cuenta con poca o ninguna documentación, y no se identifica que código se puede reutilizar, añade complejidad al desarrollo del sistema, es por eso que se necesita una arquitectura para:

- *Comprensión del sistema.* Para que una organización desarrolle un sistema, debe ser comprendido por todos los que vayan a intervenir en él. El hacer que los sistemas modernos sean comprensibles es un reto importante por muchas razones:
    a) Abarcan un comportamiento complejo
    b) Operan en entornos complejos
    c) Son tecnológicamente complejos
    d) Deben satisfacer demandas individuales y de la organización
    e) Son tan grandes que la dirección tiene que dividir el trabajo de un desarrollo en varios proyectos.
- *Organización del desarrollo.* Cuanto mayor sea la organización del proyecto de software, mayor será la sobrecarga de comunicación entre los desarrolladores para intentar coordinar sus esfuerzos. Esta sobrecarga se incrementa cuando se encuentran geográficamente dispersos. Una buena arquitectura es la que define explícitamente las interfaces haciendo que sea posible la reducción en la comunicación. Una interfaz bien definida comunica eficientemente a los desarrolladores de ambas partes que necesitan saber sobre lo que los otros equipos están haciendo. Una arquitectura y unos patrones de diseño adecuados nos ayudan a encontrar las interfaces entre los subsistemas.
- *Fomento de la reutilización.* La industria de software todavía tiene que alcanzar el nivel de estandarización que muchos dominios de hardware han conseguido, pero las buenas arquitecturas y las interfaces bien definidas son pasos en esta dirección. Una buena arquitectura ofrece a los desarrolladores un andamio sobre el cual trabajar. Un buen arquitecto

ayuda a los desarrolladores para que sepan donde buscar elementos reutilizables de manera poco costosa y para que puedan encontrar los componentes adecuados para ser utilizados.

- *Hacer evolucionar el sistema.* Si hay algo de lo que podemos estar seguros, es de que cualquier sistema de un tamaño considerable evolucionará. Incluso aunque aún esté en desarrollo. Más tarde, cuando esté en uso, el entorno cambiante provocará futuras evoluciones. Hasta que esto ocurra, el sistema debe ser fácil de modificar; esto quiere decir que los desarrolladores deberían ser capaces de modificar partes del diseño e implementación sin tener que preocuparse por los efectos inesperados que puedan tener repercusión en el sistema.

En la mayoría de los casos deberían ser capaces de implementar nuevas funcionalidades en el sistema sin tener que pensar en un impacto dramático en el diseño e implementación existentes (el sistema debe ser flexible a los cambios). Debe ser capaz de evolucionar sin problemas. Las arquitecturas de sistema pobres, por el contrario, suelen degradarse con el paso del tiempo y necesitan ser parchadas hasta que al final no es posible actualizarlas con costos razonables [Booch, *et al* 2000].

2.6.2 Pasos hacia una arquitectura

La arquitectura se desarrolla mediante iteraciones, principalmente durante la fase de elaboración. Cada iteración se desarrolla comenzando con

los requisitos y siguiendo con el análisis, diseño, implementación y pruebas, como se muestra en la Figura 2.8, pero centrándose en los casos de uso relevantes desde el punto de vista de la arquitectura y en otros requisitos [Booch, *et al* 2000].

Figura 2.8. Iteración para el proceso de desarrollo de software.

En su forma mas simple, la arquitectura es la estructura jerárquica de los componentes del programa, la manera en que los componentes interactúan y la estructura de datos que van a utilizar los componentes. Un objetivo del diseño de software es derivar una representación arquitectónica de un sistema. Ésta representación sirve como marco de trabajo desde donde se llevan a cabo actividades de diseño mas detalladas. Un conjunto de patrones arquitectónicos permiten que el ingeniero de software reutilice los conceptos a nivel de diseño.

Mientras que el tamaño y la complejidad de los sistemas de software aumenta, el problema del diseño va más allá de las estructuras de datos y algoritmos de cómputo [Garlan y Shaw, 1994]. Diseñar y especificar la estructura del sistema total emerge como una nueva clase de problema. La clave de la estructura incluye organización y estructura de control global, protocolos para comunicación, sincronización y acceso a datos, asignación de la funcionalidad de los elementos del diseño; distribución física, composición de elementos del diseño; funcionalidad y escalabilidad; y selección entre alternativas de diseño.

2.7 Simulación y modelado de escenarios

Un modelo es una representación de la realidad desarrollado con el propósito de estudiarla. En la mayoría de los análisis no es necesario considerar todos los detalles de la realidad. Entonces, el modelo no es un sustituto de la realidad sino también una simplificación de ella [Mohammad y García, 1996]. Los modelos se pueden clasificar de la siguiente forma:

a) ***Modelos iconos.*** Son los modelos físicos que se asemejan al sistema real, generalmente manejados en otra escala.

b) ***Modelos analógicos.*** Son los modelos en los que una propiedad del sistema real se puede sustituir por una propiedad diferente que se comporta de manera similar.

c) ***Modelos simbólicos.*** *S*on aquellos, en los que se utiliza un conjunto de símbolos en lugar de una entidad física para representar a la realidad. Los modelos simbólicos, dentro de los cuales se encuentran los modelos de simulación, se dividen en:

   - *Modelos determinísticos,* en estos modelos los valores de las variables no se ven afectados por variaciones aleatorias y se conocen con exactitud.
   - *Modelos estocásticos o probabilísticos,* los valores de las variables dentro de un modelo estocástico sufren modificaciones aleatorias con respecto a un valor promedio; dichas variaciones pueden ser manejadas mediante distribuciones de probabilidad.

- *Modelos dinámicos,* la característica de estos modelos es el cambio que presentan las variables en función del tempo; son ejemplos de estos modelos de series de tiempo, pronósticos y programación dinámica.
- *Modelos estáticos,* en este tipo de modelos no se maneja la variable tiempo, representan a un sistema en un punto particular.
- *Modelos continuos,* son modelos en los que las variables pueden tomar valores reales, y manejarse mediante las técnicas de optimización clásica.
- *Modelos discretos,* las variables del sistema toman valores solo en el rango de números enteros.

Independientemente de la clasificación de un modelo, existe una tendencia a seleccionarlos dependiendo de ciertas características, las cuales hacen más deseables algunos modelos que otros. Los modelos deben contar con las siguientes características:

- Confiabilidad
- Sencillez
- Bajo costo de desarrollo y operación
- Manejabilidad
- Fácil entendimiento tanto el modelo como los resultados
- La relación costo beneficio debe ser positiva [Mohammad y García, 1996].

Al modelar un sistema, se debe diferenciar entre dos tipos de datos: Los primeros permanecen sin cambio a través del tiempo y se conocen como

parámetros, los segundos presentan cambios a través del tiempo y se conocen como variables. Un modelado de un sistema es útil cuando la información del sistema tiene carácter dinámico y probabilístico

2.7.1 Ventajas y desventajas de los modelos de simulación.

Ventajas

- Una vez construido el modelo puede ser modificado de manera rápida con el fin de analizar diferentes escenarios.
- Generalmente es más barato mejorar el sistema vía simulación, que hacerlo en el sistema real.
- Es mucho más sencillo comprender y visualizar los métodos de simulación que los métodos puramente analíticos.
- Los métodos analíticos se desarrollan casi siempre, para sistemas relativamente sencillos, mientras que con los modelos de simulación es posible analizar sistemas de mayor complejidad o con mayor detalle.
- En algunos casos, la simulación es el único medio para lograr una solución.

Desventajas

- Los modelos de simulación en una computadora son costosos y requieren mucho tiempo para desarrollarse y validarse.
- Se requiere gran cantidad de corridas computacionales para encontrar soluciones óptimas lo cual repercute en el costo.

- Es difícil aceptar los modelos de simulación
- Los modelos de simulación no dan soluciones óptimas
- La solución de un modelo de simulación puede dar al analista un falso sentido de seguridad.

2.7.2 Modelado de Escenarios para Proyectos Múltiples

Miller y Lee (2004) proponen cuatro componentes básicos para la simulación de proyectos:

1 *Modelo de Escenario.* Contiene una lista de escenarios a simular. Cada escenario tiene un propósito diferente, donde el administrador de proyectos explora cosas inciertas en el desarrollo del proyecto. Los escenarios pueden clasificarse en cuatro categorías: Eventos, Políticas, Teorías y Estrategias.

2 *Modelo de Red Multi-Proyectos.* Este componente, es responsable de la ordenación en secuencia de las actividades de proyecto y la asignación de recursos utilizando el método de la cadena crítica. La red multi-proyecto también actúa como una base de datos, para almacenar la duración de actividad del proyecto que se actualizan periódicamente durante la simulación.

3 *Componente Controlador.* Es el componente que controla el sistema dinámico y la red multi-proyecto. Usando los escenarios creados en el componente modelo de escenario, el controlador coordina el curso de información entre el componente de modelo de sistema dinámico y el componente modelo de red multi- proyectos, recibe información acerca de la red de proyecto. Basado en la duración y la asignación de recursos crea el recurso apropiado para el sistema dinámico durante la simulación.

También interactúa con los proyectos individuales del modelo dinámico para asegurar las actividades que existen en la simulación dentro de los proyectos respectivos. Los resultados de la simulación serán devueltos al controlador para futuras acciones, si una cadena crítica es afectada en un proyecto por el retraso, la red puede reubicar los recursos basada en la información anterior, si este cambio de fechas afecta otro proyecto, entonces el proyecto afectado tendrá que empezar tarde, esto crea dependencias múltiples en un proyecto que afectarán la fecha de terminación del proyecto.

4 *Componente Modelo de Sistema Dinámico.* Consta de dos elementos: El modelo dinámico de recursos (MDR) y el modelo dinámico del proyecto Individual (MDRI). Estos dos modelos interactúan a través del controlador. El MDR realiza un análisis causa-efecto de los recursos que son compartidos por varios proyectos y sus relaciones de dependencia. Cada recurso representa un factor único que tendrá que ser compartido por varios proyectos. En el MDRI, se realiza un análisis causa-efecto de los

recursos en cada proyecto, dicho de otro modo, existe un análisis causa-efecto de las relaciones de dependencia entre los recursos por proyecto.

En la figura 2.9, se muestra la estructura del modelo de escenarios para proyectos múltiples propuesta por Miller y Lee.

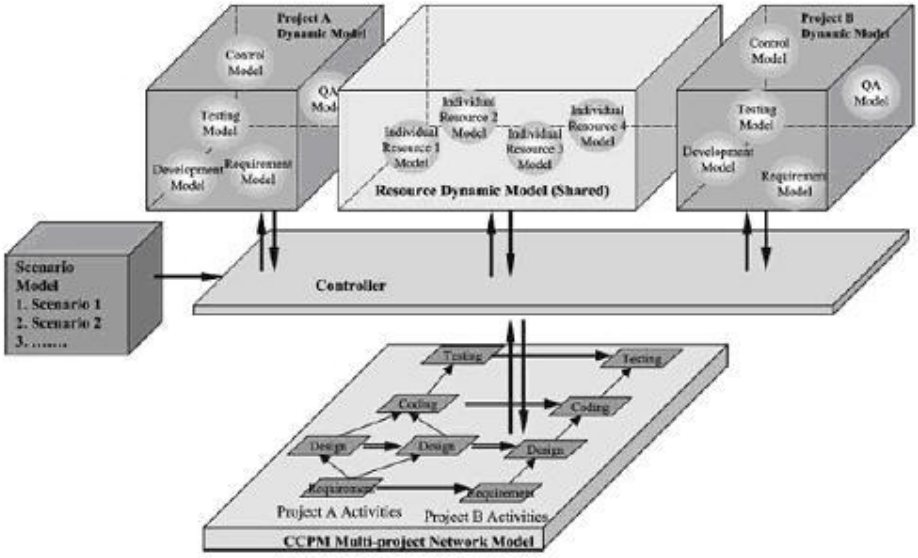

Figura 2.9. Modelado de escenarios para multi – proyectos.

2.7.3 Aplicación de la Simulación en la Calendarización de Proyectos

Padberg (2003) presentó un modelo discreto para la simulación de proyectos del software. Este modelo representa asignaciones de tareas, niveles de habilidad del personal, componentes, y la reanudación causada por diseño. Estudió sistemáticamente el funcionamiento de varias políticas para un proyecto muestra. También realizó un análisis detallado de las asignaciones de tarea que se presentan en las simulaciones. El modelo de la simulación es una

implementación del modelo de programación estocástica para proyectos de software y contiene las siguientes características:

- *Dinámica del Proyecto, e*n el modelo, el producto de software es desarrollado por varios equipos. Los equipos trabajan en paralelo, de acuerdo con un cierto diseño de alto nivel, el software se divide en componentes. En cualquier momento durante el proyecto, cada equipo trabaja con uno o más componentes y, de la misma forma, cada componente está siendo ocupado por uno o más equipos. No se requiere que varios equipos trabajen simultáneamente en todos los componentes.

  La asignación de los componentes a los equipos puede cambiar durante el proyecto. Los equipos no trabajan independientemente. Los equipos detectan de vez en cuando un problema con el diseño de alto nivel del software. Para eliminar el problema, el diseño de alto nivel se revisa. Puesto que los componentes se juntan, todos los componentes que son afectados por el cambio de diseño tendrán que ser retrabajados, no solo el componente donde fue detectado el problema.

- *Acciones de la programación de calendario,* en el modelo, un proyecto del software avanza con secuencia de fases. Una fase termina cuando el personal la hace disponible o cuando el diseño de alto nivel del software cambia.

Las acciones de la calendarización ocurren solamente al final de las fases. Las acciones programadas posibles son:

1. Asignar un componente a un equipo.
2. Un equipo comienza a trabajar con un componente.
3. Parar un equipo.

- *Estrategias,* se toma la decisión de qué equipo asignar a un componente en una fase, ésta decisión la toma el administrador del proyecto la simulación no determina ninguna estrategia.

- *Probabilidades,* el modelo de la programación de calendario es probabilístico; los acontecimientos ocurrirán solamente con cierta probabilidad en un punto en particular en un tiempo.

- *Datos de entrada,* el modelo requiere la probabilidad base (comportamiento estadístico de un equipo y un componente en proyectos anteriores) y los grados de dependencia entre componentes.

El proyecto realizado por Padberg consistió en cuatro componentes (A, B, C, D), y dos equipos (E1 y E2). El trabajo de los equipos está en paralelo. La complejidad de los componentes y la productividad de los equipos se reflejan en las distribuciones de la probabilidad que se utilizan como entrada

para las simulaciones. Las probabilidades se eligen tomando en cuenta lo siguiente:

- El equipo E2 tiene una productividad más baja que el equipo E1.
- Los componentes A y B tienen una complejidad similar.
- Los componentes C y D requieren mucho más esfuerzo que los componentes A y B.
- Los grados de la dependencia se eligen para reflejar que los componentes C y D están unidos fuertemente.

La lista de políticas contiene el orden en el que los componentes deberán ser desarrollados (prioridad). Cuando un equipo termina el componente actual, se le asigna un nuevo componente que se encuentra en la lista. La lista de políticas mantiene a los equipos ocupados todo el tiempo. En un ajuste probabilístico, los tiempos de las tareas no son conocidos con anticipación. Así una lista de prioridades no determinará en su totalidad a que equipo se le asignará un componente en particular.

Puesto que el proyecto de muestra tiene cuatro componentes (A, B, C, D) existen, 24 diferentes listas de políticas para el ejemplo del proyecto. Por ejemplo, la lista de política CDAB, inicia con el componente C asignado al equipo uno, el componente D asignado al segundo equipo. El equipo que termine la tarea primero trabajará con el componente A, y el segundo equipo en terminar la tarea trabajará con el componente B.

CAPÍTULO III. METODOLOGÍA

La investigación científica admite diversas formas de clasificarse [Zorrilla, 1987], ejemplo: básica, aplicada, documental de campo y mixta. Esta investigación utiliza la clasificación mixta, que consta de investigación de campo ó investigación directa, es decir se efectúa en el lugar y tiempo en la que ocurren los fenómenos de estudio; e investigación documental (libros, revistas, investigaciones científicas, etc.).

La investigación teórica de este trabajo comienza con la definición de administración de proyectos de software, puesto que el objetivo general de esta investigación es definir una arquitectura flexible, para un simulador de seguimiento temporal de calendario de proyectos de software con componentes intercambiables, que contemple el análisis de riesgos y el seguimiento de actividades. La información de la investigación documental y la investigación de campo recopilada y analizada fue necesaria para definir los componentes de la arquitectura propuesta.

Se buscó información acerca de metodologías para el seguimiento temporal de calendarios de proyectos, análisis de riesgos, modelos de simulación, estilos y clases de arquitecturas para el desarrollo de software, en la Figura 3.1, se muestra gráficamente la metodología que se siguió en el desarrollo de esta investigación. El proceso fue iterativo. Se inició con una búsqueda de información documental, después una investigación de campo, se analizó la información recopilada y finalmente se generó la arquitectura propuesta.

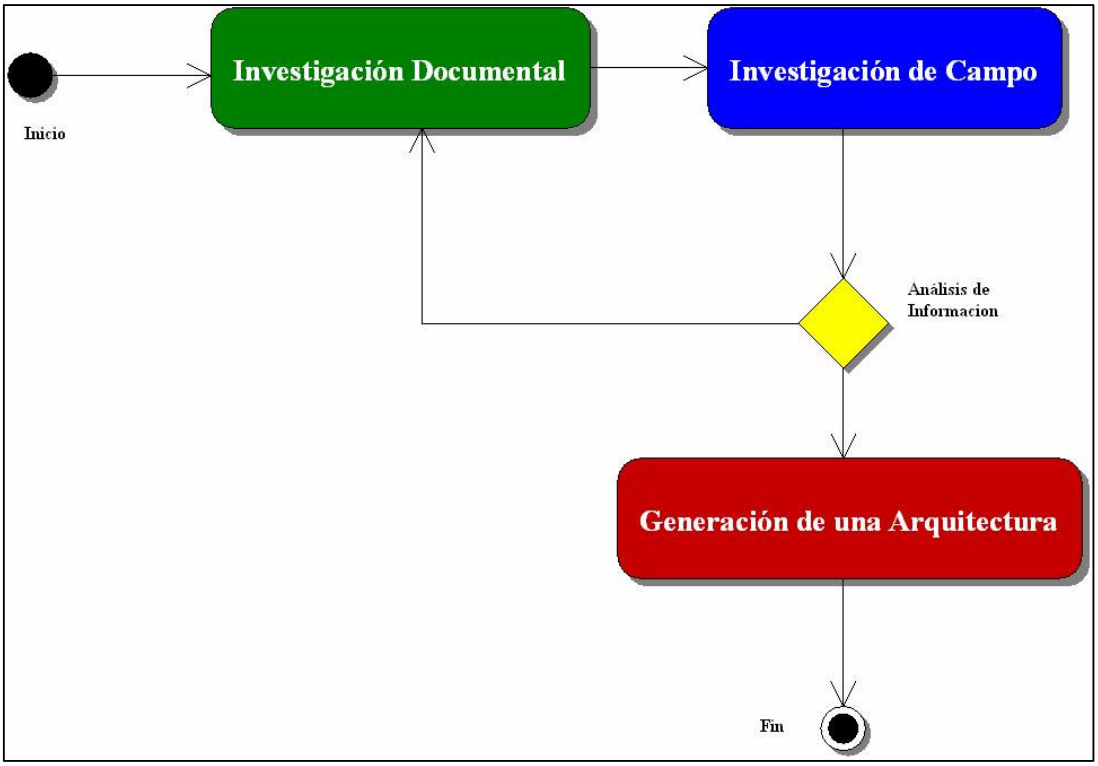

Figura 3.1. Metodología para el desarrollo de la Arquitectura.

3.1 Investigación Documental

En la actualidad existen diversos métodos para el seguimiento de calendario de actividades y administrar los riesgos en un proyecto de software, las propuestas que existen para crear un simulador de seguimiento temporal de calendario están basadas en métodos específicos de seguimiento de proyectos [Miller y Lee, 2004], [Padberg, 2003]. La investigación documental se divide en los siguientes puntos:

1) *Elección del problema.* Se selecciona el problema a resolver en la investigación. Esta etapa se descompone en las siguientes elecciones sucesivas para formar el sustento de la investigación documental del proyecto:

   a. Elección del campo de Investigación
   b. Elección de métodos y técnicas de trabajo
   c. Elección de los tópicos de investigación

2) *Planeación del Trabajo.* La planeación del trabajo fue un proceso iterativo apoyado en la gráfica de Gantt para el seguimiento de actividades a través del tiempo. Se realizaron las siguientes actividades:

   a. Bibliografía provisional
   b. Definición del problema
   c. Formulación de la programación de trabajo

3) *Acopio de Información.* Se recopiló la información del marco teórico de la investigación que sustenta el desarrollo del proyecto.

4) *Interpretación de la información.* Se realizó un análisis de la información recopilada y se elaboró una síntesis de la información más importante.

5) *Redacción del marco teórico.* Ordenar el texto resultado de la síntesis de ideas encontradas en el punto anterior.

3.2 Investigación de Campo

Se llevaron a cabo entrevistas informales del tipo no estructurada con administradores de proyectos, de tres empresas de software, generando preguntas abiertas acerca del ambiente que se vive en el desarrollo de software, dando completa libertad al entrevistado de mostrar su punto de vista y experiencia profesional. Estas entrevistas se realizaron en forma de conversación tomando la modalidad de entrevista clínica, para no incomodar a los entrevistados ya que hay un completo hermetismo sobre el desarrollo de sus proyectos.

La primera etapa de las entrevistas se realizó al inicio del proyecto. Durante el mes de Diciembre del 2004, se visitaron a 9 empresas de software durante tres días, y solo una compañía accedió a dar una entrevista previo a una cita (en el futuro se mencionará como Empresa 1).

La segunda etapa de las entrevistas se realizó entre los meses de septiembre y octubre del 2005. Se hizo una primera entrevista a la empresa de software (Empresa 2), donde se definió la programación para las entrevistas, los temas a discutir así como el tiempo estimado de la entrevista. Se realizó una entrevista de 40 minutos cada 15 días durante dos meses con el administrador del proyecto. En la tercera etapa de las entrevistas se hizo una estancia de aprendizaje en la Empresa 3, con 4 horas diarias de lunes a viernes durante un mes.

Los temas tratados durante las entrevistas con los administradores de proyectos de las tres empresas fueron los siguientes:

- Estructura de la empresa
- Magnitud de proyectos de software de la compañía
- Procesos para el desarrollo de software
- Metodologías para la administración de actividades
- Gestión de riesgos
- Cambios en los Requerimientos
- Casos de éxito en sus proyectos
- Principales causas por las que los proyectos no terminan en tiempo (Casos fallidos)

En la Empresa 2 se analizaron dos proyectos: uno que no terminó en tiempo, y uno que no se realizó. En los dos casos ocurrió un acontecimiento no deseable y no administrado o planeado y esto ocasionó que los tiempos y los costos del proyecto se salieran de lo planeado. Dentro de la estancia de aprendizaje en la Empresa 3, se observó que no había un proceso definido para el desarrollo de software. Esta empresa apenas empezaba a visualizar la importancia de los procesos en el desarrollo de software, después de haber iniciado sus labores nueve años atrás. Tampoco manejaba un análisis de riesgos en el proyecto, y el cambio en los requerimientos era frecuente, en consecuencia los proyectos no terminaban en tiempo y algunos en ocasiones después de haberlos liberado se les encontraban errores graves por parte de sus clientes, por ejemplo: falta de componentes, o componentes inconclusos,

incompatibilidad con el sistema, fallas de captura etc. De las entrevistas con los administradores de proyectos, se encontró una enorme fuente de oportunidad al observar, que en muchos casos los proyectos se retrasaban por una inadecuada planificación de las actividades en el proyecto y por no hacer un análisis de los riesgos implícitos en el proyecto.

En la Empresa 2 se hacia un análisis de riesgos pasivo, es decir no se atacaba el riesgo antes de iniciar el proyecto, solo se identificaba y solo si aparecía durante el desarrollo, se trataba de solucionarlo y en algunos caso posponer su solución hasta que fuera inevitable su afectación en el calendario.

En la Empresa 3 se acostumbraba no hacer un análisis de los posibles riesgos que podían ocurrir en el proyecto. No existía un seguimiento de actividades establecido, tampoco estaba definido que hacer en caso de que ocurriera algún evento no deseado en el proyecto. El administrador de proyectos simplemente esperaba que no ocurriera algún contratiempo que pusiera en peligro la fecha de terminación del proyecto, y no hacia una estimación del tiempo que tardaría en corregir ese problema si apareciera, tras nueve años de haber iniciado sus actividades, apenas empezaban a realizar bitácoras de tiempos y actividades para programaciones futuras, es decir, para que en el próximo proyecto tomar en cuenta los tiempos muertos y realizar una estimación más acertada del tiempo de terminación del proyecto, en vez de enfocar sus esfuerzos en reducir esos tiempos creados por falta de planeación y de no realizar una oportuna gestión de riesgos en sus proyectos.

Al inicio de este proyecto se analizaron diferentes métodos para la administración de proyectos y gestión de riesgos, para adoptar una que fuera sencilla de implementar y eficiente para realizar el seguimiento de las actividades y realizar la gestión de riesgos en el proyecto. Durante las entrevistas con las empresas 1 y 2 y la estancia de aprendizaje que se realizó en la empresa 3, se observó que utilizan diferentes métodos para el seguimiento de las actividades y para la gestión de riesgos y en ocasiones los métodos utilizados pueden variar de un proyecto a otro dependiendo su magnitud en la misma empresa. De tal modo, es necesario tomar en cuenta, que los métodos para el seguimiento de calendario de actividades y la gestión de riesgos, son dos variables más que se agregan a la complicada labor de la administración de proyectos de software.

Con la información obtenida de las entrevistas y recopilada en investigación documental, se establecieron los requisitos de la arquitectura, se crearan los casos de uso y escenarios.

3.3 Generación de una arquitectura

El proceso para la creación de la arquitectura de un simulador de seguimiento temporal de calendario de proyectos de software fue iterativo y esta basado en los lineamientos del RUP.

CAPÍTULO IV. ARQUITECTURA DEL SIMULADOR

En base al análisis teórico y práctico que se realizó durante el desarrollo de la tesis, se propone una arquitectura de un simulador de seguimiento temporal de calendario de proyectos de software que contenga componentes independientes e intercambiables, creando así una arquitectura modular que permita identificar las relaciones entre componentes así como sus conexiones. Ésta arquitectura deberá ser capaz de incorporar nuevas funcionalidades en el sistema, sin que cause un gran impacto en el diseño o la implementación.

La arquitectura debe ser flexible a los cambios entre componentes y capaz de evolucionar sin mayor problema, los componentes podrán ser removidos e intercambiados por otros componentes cuando sus interfaces de comunicación contemplan con los métodos establecidos en la arquitectura. Este comportamiento debe ser transparente para el usuario.

A continuación, se explican los casos de uso así como los diagramas de secuencias que cumplen con las restricciones de este proyecto. Cabe mencionar que las restricciones para la creación de esta arquitectura se basan en las diferentes metodologías para el seguimiento de actividades y el análisis de riesgos que se analizaron a través de la exploración de campo y recopilación teórica.

4.1 Casos de Uso

Para fines prácticos, la arquitectura propuesta se basa en dos casos de uso en los que interviene un solo actor. En el escenario, el administrador de proyectos, realiza un establecimiento y la simulación del proyecto de manera exitosa.

La Figura 4.1, muestra el diagrama de casos de uso. El actor es el administrador de proyectos y los casos de uso son *Establecer Proyecto y Simular Proyecto.*

Caso de uso *"Establecer Proyecto"*. El administrador de proyectos establece el proyecto que posteriormente será simulado, introduce los datos que se requieren para la administración del proyecto de forma satisfactoria, ejemplo: actividades a realizar, recursos con que cuenta (humano, infraestructura), estimación de tiempos (fecha de inicio, fecha de terminación), prioridades (actividades que requieren mayor atención o iniciar de manera temprana, interdependencia entre recursos, etc.)

Caso de uso *"Simular Proyecto".* El administrador de proyectos después de establecer el proyecto, inicia la simulación y ésta cuando se termina mostrará los resultados.

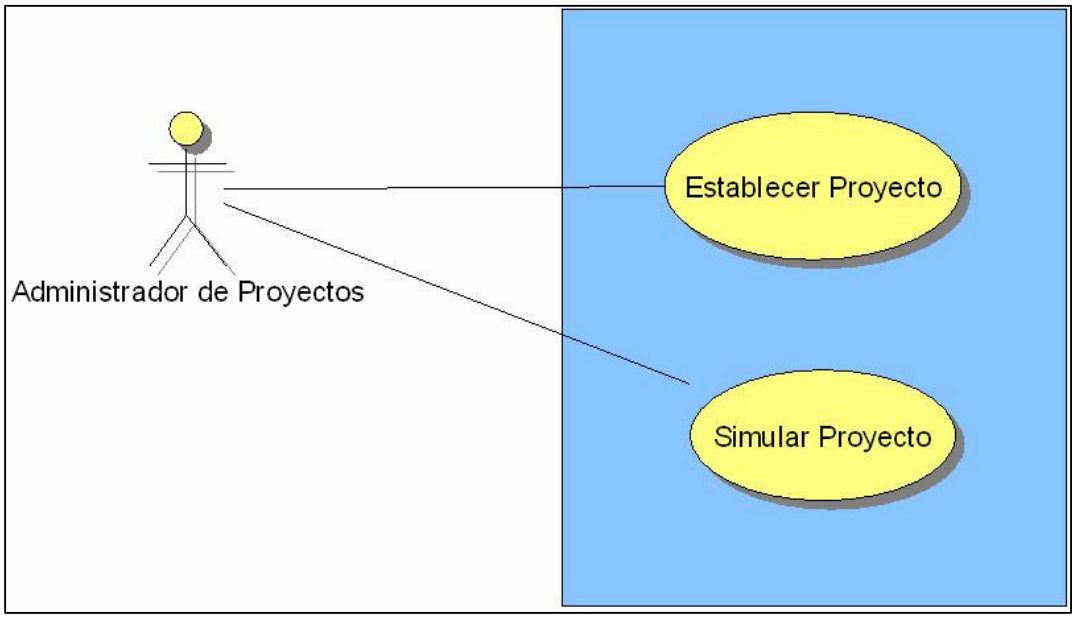

Figura 4.1. Diagrama de Casos de Uso.

Para implementar de la arquitectura propuesta, es necesario considerar escenarios en donde existan complicaciones en establecimiento y la simulación del proyecto de software. Estos escenarios tendrán que ser explicados a detalle con los diagramas que se generen para la creación del simulador.

4.2 Modelado de Escenarios con Diagramas de Secuencia

Se realizó un diagrama de secuencia para cada caso de uso (establecer proyecto y simular proyecto), en éste se describe el escenario principal para cada caso.

4.2.1 Diagrama de secuencia para el caso de uso establecer proyecto

El administrador de proyectos a través de la interfaz de usuario realiza una petición al controlador de componentes para establecer un nuevo proyecto. Los datos que el administrador proporciona son almacenados en una base de datos. La figura 4.2, muestra el diagrama de secuencias para el caso de uso establecer proyecto.

Este caso de uso lo inicia el administrador de proyectos en la *Interfaz Captura*, en donde el actor, realiza el establecimiento del proyecto mediante la captura de información. Esta información es dirigida hacia el controlador de componentes que a su vez envía la información al conector a base de datos para que deposite la información del proyecto en un administrador de bases de datos.

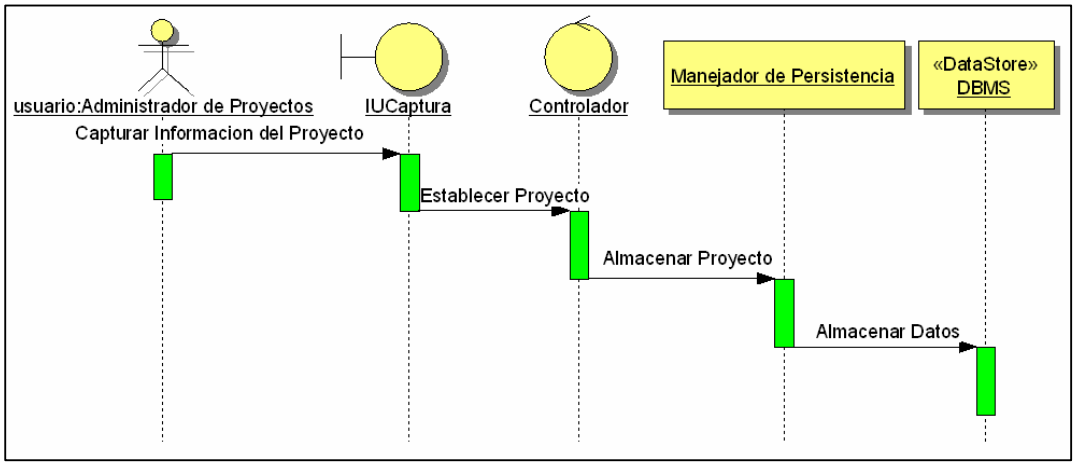

Figura 4.2. Diagrama de secuencia para el caso de uso establecer proyecto.

4.2.2 Diagrama de secuencia para el caso de uso simular proyecto

Cuando el administrador de proyectos realiza una petición al SiPS (Simulador de proyectos de Software), para realizar una simulación de un proyecto, el *Controlador de Componentes* alerta al *Componente Simulador* para que prepare la simulación del proyecto antes establecido, el *Componente Simulador* realiza una petición al *Controlador de Componentes* y quien realiza una petición al componente *Manejador de Persistencia* para recuperar el proyecto depositado en la base de datos. El *Manejador de Persistencia* se comunica con la base de datos recuperando la información que le fue pedida y la envía de regreso para que el *Componente Simulador* tenga la información del proyecto.

Una vez que los datos se encuentran en el simulador, éste se prepara para iniciar la simulación. El *Componente de Riesgos* y el *Componente de Red* son advertidos de que inicia la simulación del proyecto. El *Componente Simulador* manda información al *Componente de Riesgos* y al *Componente de Red*. El *Componente de Red* le regresa la red de actividades en el proyecto y el *Componente de Riesgos* el análisis de riesgos en el proyecto. Estos datos son enviados del *Componente Simulador* al *Componente Historial* para crear una bitácora del proyecto. El *Componente Simulador* empieza la simulación y se determina si existirá un riesgo en el proyecto en un intervalo de tiempo. Cuando se activa un riesgo en la simulación del proyecto, el *Componente Simulador* manda la información al *Componente de Riesgos* quien determina el impacto de ese riesgo en todas las actividades relacionadas en ese momento, y regresa

la información al *Componente Simulador* que a su vez la envía al *Componente de Red*. El *Componente de Red* genera la nueva red de actividades y la envía al *Componente Simulador*, el *Componente Simulador* envía esa información al *Componente Historial* para que guarde los cambios en el proyecto, después el *Controlador de Componentes* manda un mensaje a la *Interfaz de Usuario* indicando que se ha modificado la red de actividades en el proyecto.

Este proceso se repite hasta que el proyecto llega a su fin, cuando la simulación termina el *Controlador de Componentes* envía la información de la simulación del proyecto a la *Interfaz de Usuario*. En la figura 4.3, se muestra el diagrama de secuencias para el caso de uso simula proyecto, en el que se observa las relaciones entre los objetos, y la secuencia de mensajes que se envían de un objeto a otro.

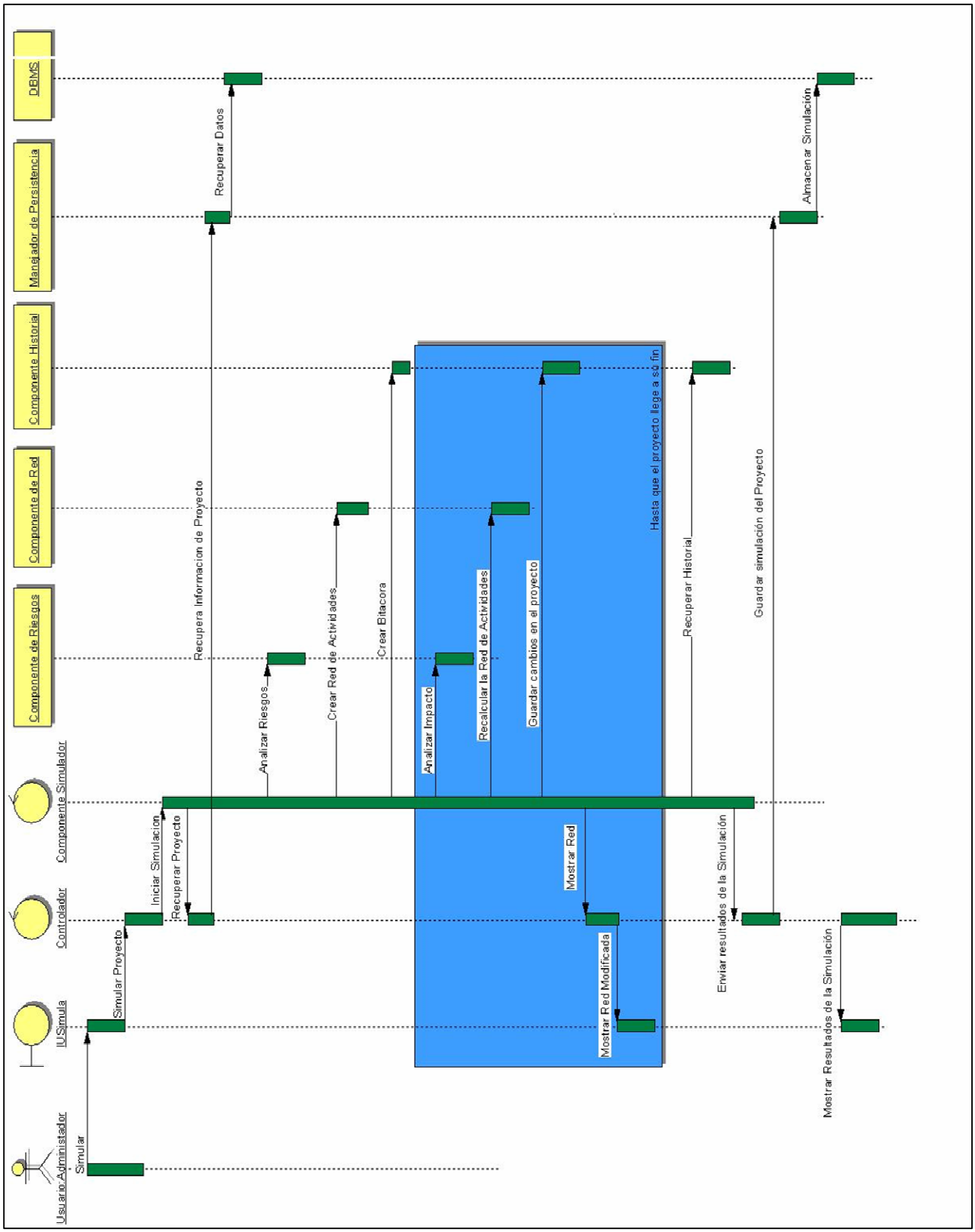

Figura 4.3. Diagrama de secuencias caso de uso simular proyecto.

4.3 Diagrama de Componentes

La vista de implementación se representa en diagramas de componentes, el diagrama de componentes muestra los tipos de componentes del sistema y sus dependencias.

Como resultado del análisis de los casos de uso, escenarios y retomando la información recabada en la investigación de campo, la arquitectura propuesta consta de los siguientes componentes:

- *Componente de Red*
- *Componente de Riesgos*
- *Componente Historial*
- *Componente Simulador*
- *Controlador de Componentes*
- *Manejador de Persistencia*
- *Interfaz de Usuario*

La Figura 4.4, muestra la arquitectura propuesta del simulador. Los componentes contienen interfaces mediante las cuales los componentes se comunican e interactúan entre sí.

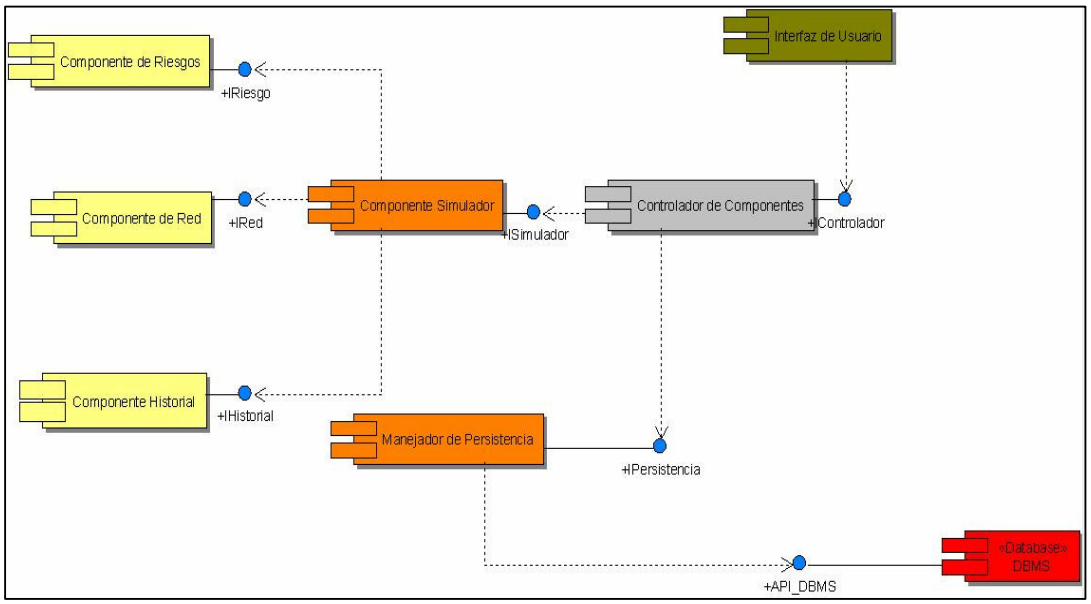

Figura 4.4. Diagrama de Componentes.

4.4 Diagramas de Clases

Los elementos principales de la vista estática son el diagrama de clases y sus relaciones. A continuación se detallan cada uno de los componentes propuestos y las clases que realiza:

1. **Interfaz de Usuario.** Por medio del *Componente Interfaz* de Usuario el administrador de proyectos de software se comunica para dar de alta un nuevo proyecto o realizar una simulación de un proyecto ya existente en la base de datos. La clase Interfaz de Usuario hereda a las clases IUActividad, IURiesgo, IURecursos y IUSimulacion, los atributos (ejemplo; nombre de la actividad, nombre del recurso) y métodos (Ejemplo capturar, enviar,

mostrar) necesarios para la implementación. Los métodos principales de esta clase son EstablecerProyecto y SimularProyecto, tal como se muestra en la Figura 4.5.

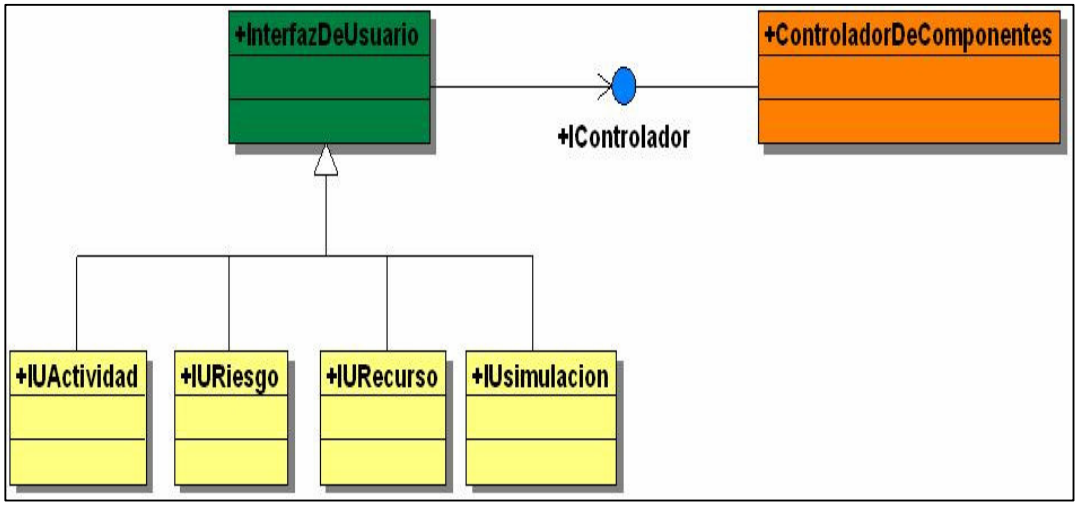

Figura 4.5. Diagrama de clases de la interfaz de Usuario.

IUActividad. Esta clase tiene la responsabilidad de comunicar al usuario para que de de alta, modifique o elimine una actividad en el proyecto

IURiesgo. Permite al usuario registrar un riesgo en el proyecto.

IURecurso. A través de esta interfaz, el usuario asigna los recursos a cada una de las actividades en el proyecto.

IUSimulacion. El usuario a través de esta interfaz inicia la simulación de un proyecto existente.

Los métodos principales de esta clase son: Establecer Proyecto y Simular Proyecto

2. **Manejador de Persistencia.** Este componente es el encargado de comunicarse con la base de datos. Por medio de este componente viaja la información del *Controlador de Componentes* a la base de datos y de la

base de datos al *Controlador de Componentes*. La información que es capturada en la *Interfaz de Usuario* (actividades, recursos, riesgos, etc.), es enviada a través del *Manejador de Persistencia* para almacenarse temporalmente y después ser recuperada cuando se inicie la simulación. Este componente se comunica con la base de datos a través de la interfaz API_DBMS, como se muestra en la Figura 4.6. Los principales métodos que maneja esta clase son: RecuperarDatos, AlmacenarDatos y AlmacenarSimulacion

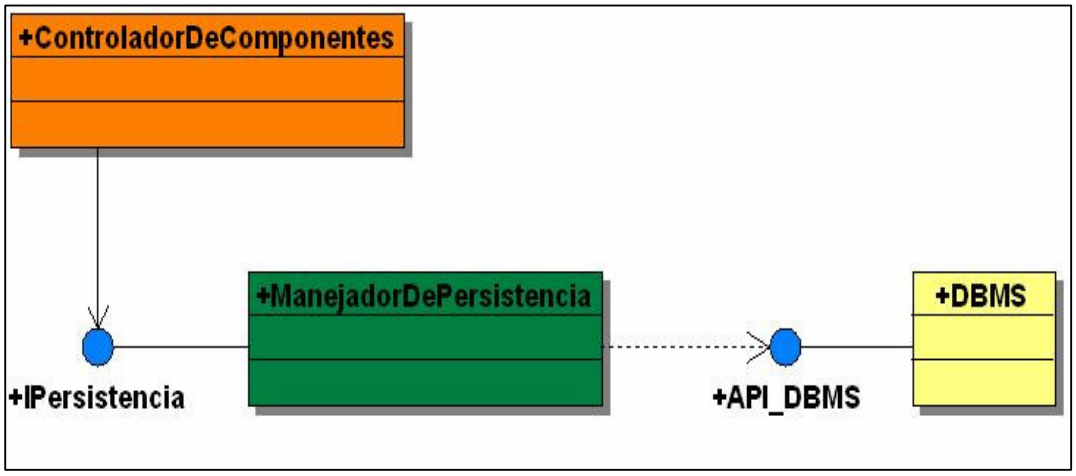

Figura 4.6, Diagrama de clases del Manejador de Persistencia.

3. **Componente de Red:** El C*omponente de Red*, es uno de los componentes principales para el seguimiento temporal de actividad, es en el proyecto, administra los tiempos en el proyecto determina que actividad le antecede y sigue a cada actividad, administra los tiempos de duración por actividad y los tiempos de protección en el proyecto, generando una red de actividades.

El *Componente de Red* tiene constante comunicación con el *Componente Simulador*. El *Componente Simulador* realiza una petición de

servicio al *Componente de Red* para que cree la red de actividades inicial en el proyecto, posteriormente cuando se genera un riesgo el *Componente Simulador* envía esa información al *Componente de Red* para que éste recalcule la red de actividades en el proyecto. *El Componente de Red* se comunica con el simulador a través de la interfaz *IRed*.

En la Figura 4.7, se muestra el diagrama de clases RedDeActividades, que tiene relación con las clases, *Calendario*, *Actividad* y *Recurso* para poder generar la red de actividades del proyecto de software.

Los métodos principales de la clase RedDeActividades son: CrearRedDeActividades, EnviarRedDeActividades, RecalcularRedDeActividades y EnviarRedRecalculada.

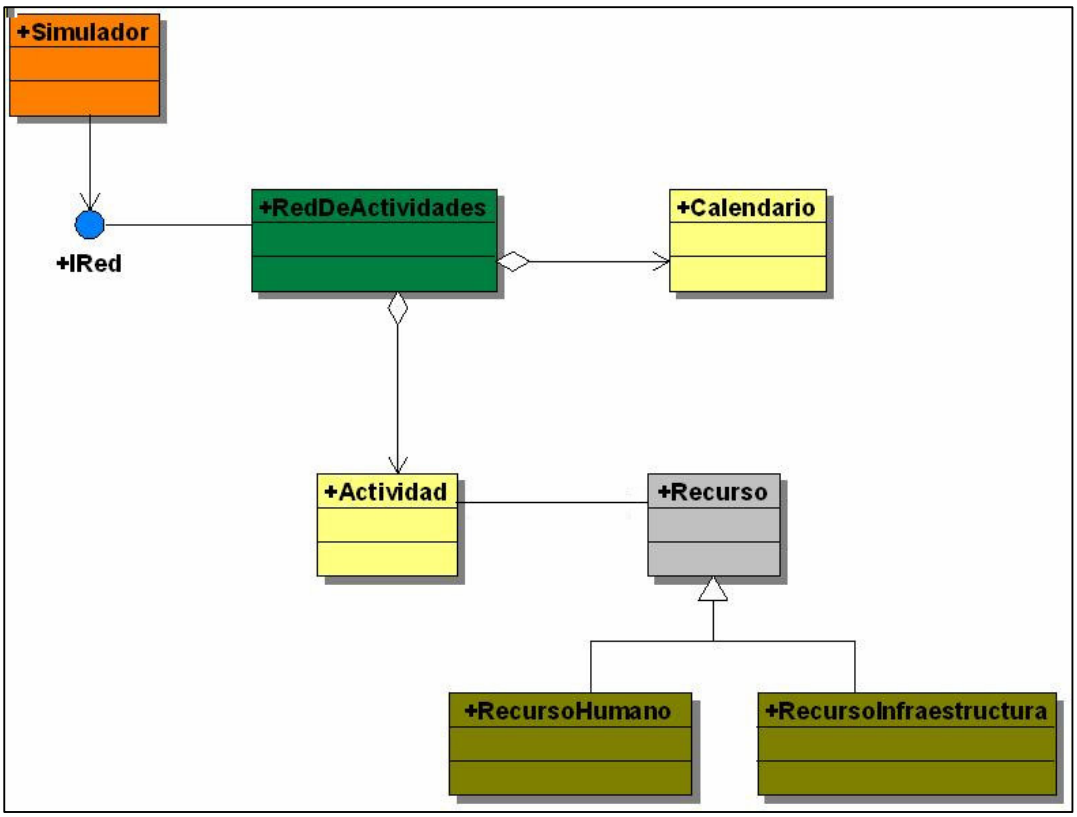

Figura 4.7. Clases del Componente de Red.

4. **Componente de Riesgos:** El *Componente de Riesgos*, realiza un análisis de riesgos y un análisis de impacto de los posibles riesgos relacionados con cada actividad del proyecto. El *Componente Simulador* realiza una petición de servicio al *Componente de Riesgos* y envía la información relacionada con las actividades del proyecto, para que el *Componente de Riesgos* genere una lista de riesgos priorizados en el proyecto. *El Componente de Riesgos* identifica el riesgo en el proyecto y de acuerdo con la probabilidad de aparición y el impacto que pudiera causar a las actividades relacionadas realiza una priorización del riesgo. Cuando el *Componente Simulador* activa un riesgo en un tiempo *t* de la simulación, envía una petición al *Componente de Riesgos* para que determine el grado de impacto del riesgo activado en todas las actividades relacionadas en ese tiempo *t*. Una vez que se analiza el impacto por actividad, la información es enviada al *Componente de Red* a través del *Componente Simulador* para que recalcule la red de actividades. El *Componente de Riesgos* cuenta con la interfaz IRiesgo por medio de la cual tiene una comunicación frecuente con el *Componente Simulador*. En la Figura 4.8, se muestran las clases del *Componente de Riesgos*, en donde a la clase AdministradorDeRiesgo se le agrega uno o varios riesgos que pueden depender de una actividad o la complejidad del proyecto.

El *Componente de* Riesgos puede ser reemplazado por otro componente que realice la misma función, siempre y cuando incluya en la interfaz de comunicación los métodos necesarios para interactuar con el

componente simulador. Esto se debe a que existen diferentes métodos para gestionar riesgos en un proyecto de software.

La clase AdministradorDeRiesgos cuenta con los siguientes métodos principales: Administrador de Riesgos son: AnalizarRiesgo, AnalizarImpacto, PriorizarRiesgo, EnviarAnálisisDeRiesgo, EnviarAnálisisdeImpacto.

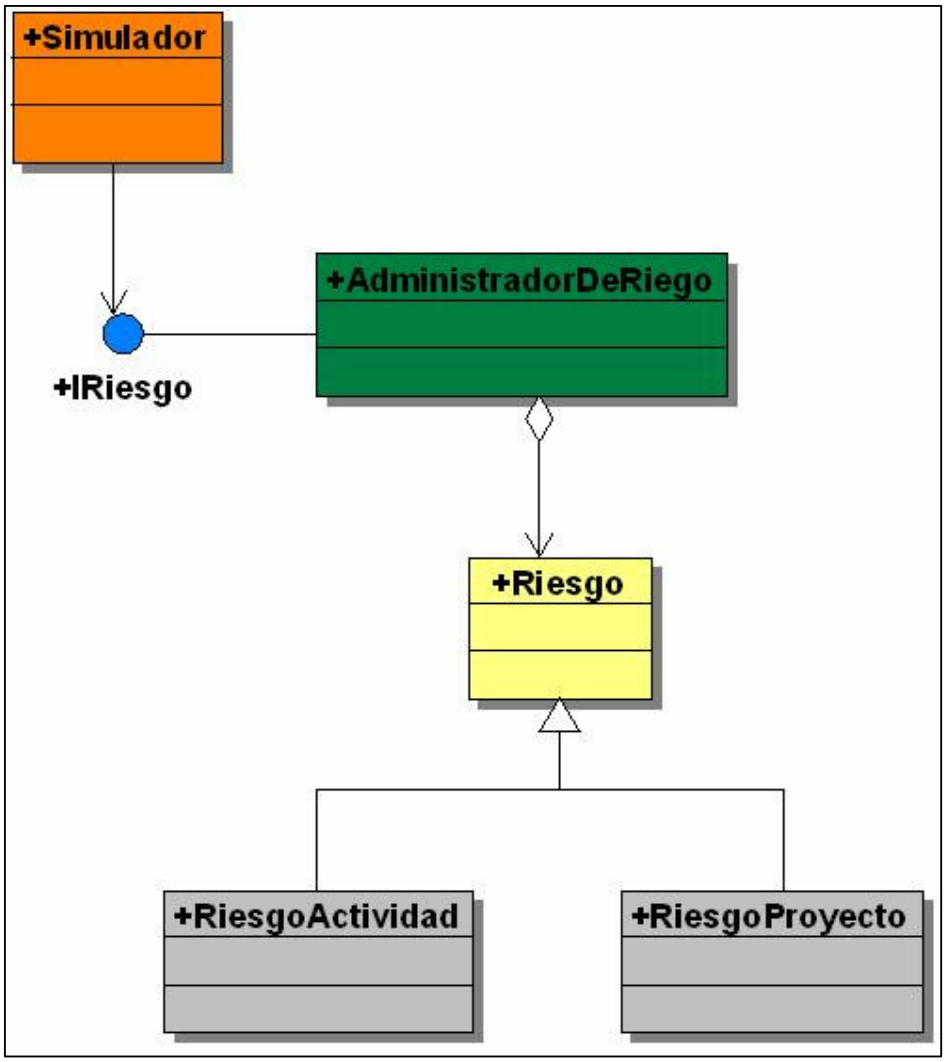

Figura 4.8. Clases del Componente de Riesgos.

5. **Componente Historial.** El *Componente Historial*, lleva una bitácora de la simulación del proyecto a través del tiempo, desde que el proyecto inicia hasta que termina. Se comunica con el *Componente Simulador* a través de la Interfaz Historial. El *Componente Simulador* envía al *Componente Historial* la información relacionada con la red y los riesgos en el proyecto. Cada variación en el proyecto es guardada en el *Componente Historial* creando una fotografía del proyecto a través del tiempo. Cuando la simulación termina esta información es enviada a la Interfaz de Usuario para que el administrador de proyectos visualice las modificaciones que sufrió el proyecto durante la simulación. En la Figura 4.9, se muestran las clases que están relacionadas con el *Componente Historial*, Los métodos principales que utiliza la clase Historial son: CrearBitácora de Proyecto, GuardarCambiosEnElProyecto y EnviarHistorial.

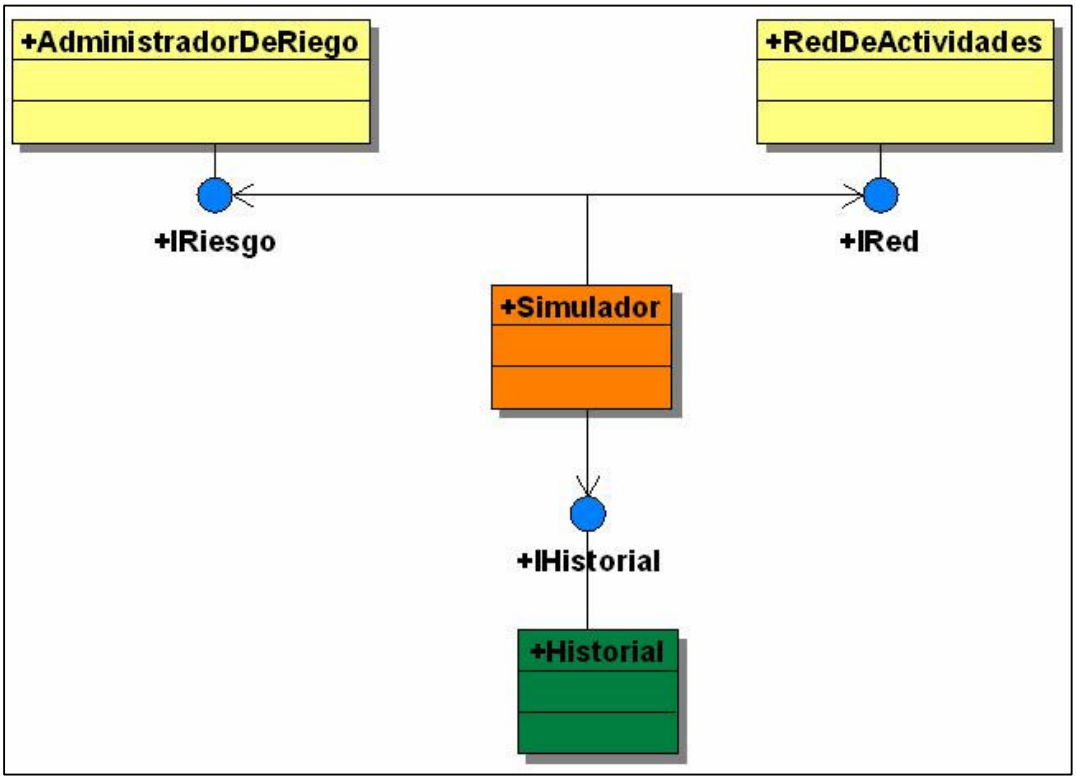

Figura 4.9. Clases del componente Historial.

6. **Componente Simulador.** Es el componente que lleva el control en la simulación e interactúa con la mayoría de los componentes en el sistema. El *Componente Simulador* se comunica con el *Componente de Red*, para que se cree la red de actividades del proyecto. Se comunica con el Componente *de Riesgos* para que se analicen y prioricen los riesgos para las actividades del proyecto. El *Componente Simulador* también se comunica con el *Componente Historial* cuando crea la bitácora de la simulación. Coordina la comunicación de peticiones, respuestas y excepciones entre los componentes mientras la simulación se está llevando a cabo. Se comunica con el *Controlador de Componentes* cuando se lleva a cabo el inicio de simulación y cuando la simulación termina.

En cada iteración del proyecto el *Componente Simulador* se comunica con todos los componentes que intervienen en la simulación. El *Componente* Historial es el encargado de llevar todas las modificaciones que se realizan mientras la simulación se realiza. El *Componente de Red*, genera la red de actividades inicial del proyecto y las redes de actividades que son modificadas por la actividad de un riesgo en la simulación. Cada vez que se activa un riesgo en el proyecto, el *Componente de Riesgos* tiene que realizar un análisis de impacto para cada actividad relacionada con ese riesgo en un tiempo. El *Componente de Riesgos*, envía la información al Componente Simulador con todas las actividades que se vieron afectadas al momento de activarse el riesgo. La comunicación entre clases del *Componente Simulador* se realiza mediante la interfaz ISimulador. En la Figura 4.10, se muestran las clases que utiliza el *Componente Simulador*

para llevar a cabo la simulación del proyecto de software. Los métodos más importantes de la clase Simulador son los siguientes: IniciarSimulacion, RecuperarProyecto, AnalizarRiesgos, CrarRedDeActividades, CrearBitacora, AnalizarImpacto, RecalcularlaRedDeActividades, GuardarLosCambiosEnElProyecto, MostrarRed, RecuperarHistarial, EnviarResulatadosDeLaSimulacion.

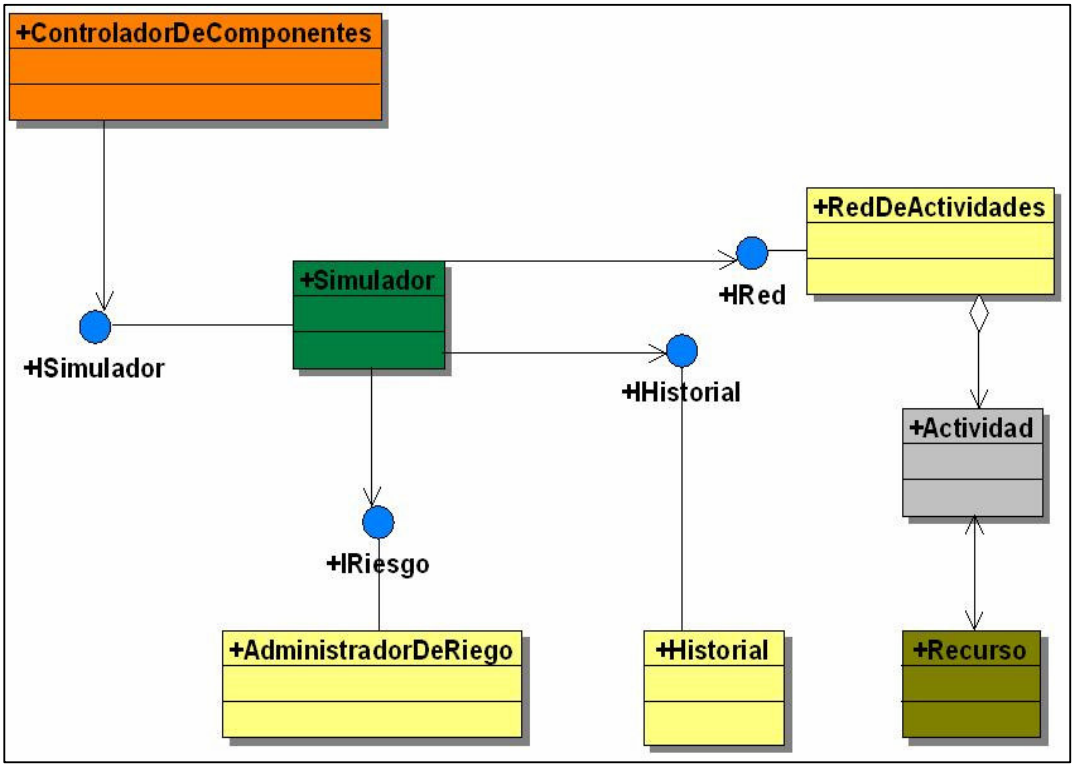

Figura 4.10. Clases del componente Simulador.

Nota: En el Anexo A se encuentra una propuesta para crear un componente simulador Basado en riesgos.

7. **Controlador de Componentes.** El *Controlador de Componentes*, filtra datos e información que llegan de la *Interfaz de Usuario*, se comunica con el *Componente Manejador de Persistencia* y el *Componente*

Simulador. El *Controlador de Componentes* controla el transito de información de la *Interfaz de Usuario* al *Componente Manejador de Persistencia*, y del *Manejador de Persistencia* al *Componente Simulador.* El *Controlador de Componentes* es responsable de avisar al *Componente Simulador* cuando debe de iniciar la simulación, el *Componente Simulador* se prepara y realiza una petición para que se recupere la información del proyecto a simular. El *Controlador de Componentes*, envía la información del proyecto que es recuperada por el *Manejador de Persistencia*. El *Controlador de Componentes* se comunica con la *Interfaz de Usuario*, a través de la Interfaz IControlador. Cuando finaliza la simulación el *Componente Simulador* envía la bitácora de la simulación del proyecto al *Controlador de Componentes* para ser enviada a la *Interfaz de Usuario* y así el administrador analice la información del proyecto simulado. Los principales métodos de la clase Controlador son los siguientes: AlmacenarProyecto, IniciarSimulacion, RecuperarInformacionDelProyecto, MostrarRedModificada, GuardarSimulacionDelProyecto, MostrarResultadosDeLaSimulacion. En la Figura 4.11 se muestran las clases que están relacionadas con el controlador de componente así como la interfaz que utiliza para la comunicación.

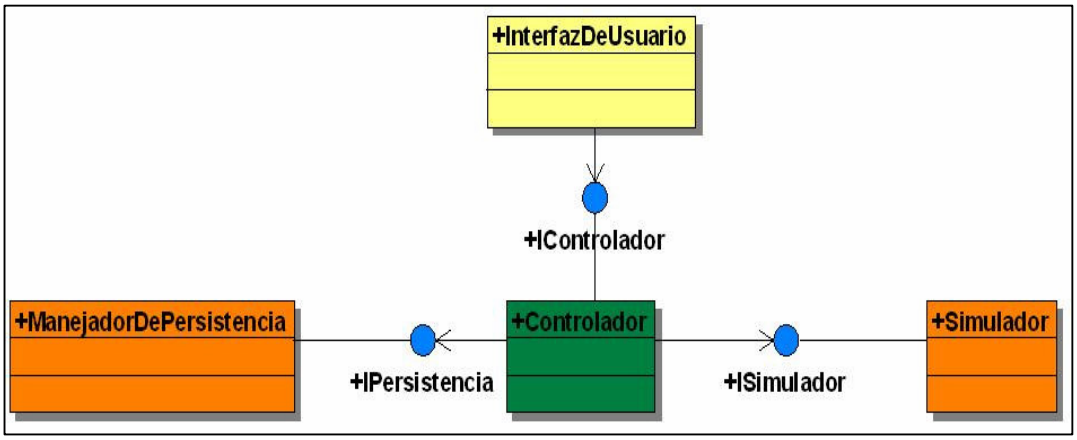

Figura 4.11. Clases del Controlador de Componentes.

CAPÍTULO V. CONCLUSIONES

5.1 Conclusiones

Existe incertidumbre en la administración de proyectos de software, debido a que hay innumerables variables que pueden afectar las fechas de entrega. El administrador del proyecto debe tomar en cuenta diversos factores que influyen directa o indirectamente en la fecha de entrega de su producto, cambio de los requerimientos, falta de capacitación, rotación de personal etc. Los proyectos de software tienden a no terminar en tiempo por una mala administración de las actividades en el proyecto y por el desinterés de administrar los posibles riesgos en el proyecto de algunas empresas de software. Esto fue el motivo principal para realizar éste proyecto.

De las entrevistas que se realizaron con las empresas de software y la estancia de aprendizaje en una de ellas, se puede concluir que los principales problemas con los que se enfrentaron los administradores de proyectos, y por

consecuencia ocasionaron perdidas económicas y retrasos en las fechas establecidas de entrega de sus proyectos fueron: una inadecuada planificación de las actividades en el proyecto, cambios de requerimientos en el proyecto y la evasión de análisis en riesgos latentes.

Los administradores de proyectos necesitan una herramienta para planear actividades y administrar riesgos bajo un ambiente simulado en el que puedan representar proyectos reales con sus dificultades, sin tener que sufrir las pérdidas económicas y el tiempo que ocasiona administrar proyectos reales.

La arquitectura propuesta permite construir un simulador de seguimiento temporal de calendario de proyectos de software flexible y reutilizable, bajo un enfoque de análisis de riesgos y un modelo para la administración de proyectos de software. La arquitectura propuesta servirá como modelo para la construcción de una herramienta que auxilie al administrador de proyectos de software.

5.2 Recomendaciones para Trabajo a Futuro

Este documento propone una arquitectura para la creación de un simulador de proyectos de software, en donde el escenario contempla el seguimiento temporal de actividades de un proyecto, teniendo como base la gestión de riesgos en el proyecto. Para trabajos futuros se podría implementar lo siguiente en la arquitectura:

- ***Un prototipo de un simulador de seguimiento temporal de actividades.*** Se podría crear un prototipo de un simulador siguiendo la arquitectura propuesta para realizar simulaciones de proyectos de software reales y analizar los resultados.
- ***La retroalimentación entre el administrador de proyectos y sistema durante la simulación.*** Una mejora en esta arquitectura, sería implementar el escenario donde el administrador de proyectos modifique los requerimientos del sistema, incrementar o disminuir el número de actividades, recursos y/o tiempos asignados.
- **La simulación de seguimiento temporal de actividades con proyectos múltiples.** El administrador de proyectos podrá elegir la opción de simular un solo proyecto o más de uno en un tiempo determinado.
- ***Un componente de ayuda para el usuario.*** Este componente seria una especie de tutor para el administrador de proyectos y fungiría como un entrenador para futuros administradores.

REFERENCIAS

[Acosta, 1997], Acosta, J. 1997. “Teoría de decisiones”. Representación y servicios de Ingeniería, S. A. México.

[Anderson, *et al,* 1999], Anderson, Sweeney, Williams. 1999. “Métodos Cuantitativos para los Negocios”. International Thomson Editores, México,

[Boehm, 1991], B. W. Boehm. jan.1991. “Software risk management: principles and practices” .IEEE software, pp 32-41

[Booch, *et al,* 2000], James Rumbaugh, Ivar Jacobson, Grady Booch. 2000. “El Proceso Unificado de Desarrollo de Software”. Addison Wesley.

[Booch, *et al,* 2004], James Rumbaugh, Ivar Jacobson, Grady Booch. 2004. “Alalysis and Design for the .NET Plataform, Practical Software Engineering”. Addison Wesley.

[Booch, *et al,* 1999], James Rumbaugh, Ivar Jacobson, Grady Booch. 1999. “The Unified Modeling Language User Guide, UML”. Addison Wesley.

[Booch, *et al,* 1999b], James Rumbaugh, Ivar Jacobson, Grady Booch. 1999. “The Unified Modeling Language Reference Manual, UML”. Addison Wesley.

[Cho y Eppinger, 2005], Soo-Haeng Cho and Steven D. “A Simulation-Based Process Model for Managing Complex Design Projects”, Eppinger*, Member, IEEE vol,52, No.3 August 2005*

[Garlan y Shaw, 1994], David Garlan and Mary Shaw January 1994. “An Introduction to Software Architecture” Also appears as CMU Software Engineering Institute Technical Report. CMU/SEI-94-TR-21, ESC -TR-94-21.

[Garlan y Shaw, 2002], David Garlan and Mary Shaw January, 2002. “Formulations and Formalisms in Software Architecture”, Volume 1000-Lecture Notes in Computer Science, Springer Verlag, 1995.

[Goldratt, 2000], Goldratt, Eliyahu. 2000. “Cadena Crítica”, Ediciones Castillo, México

[Lawrence, 2002], Shari Lawrence Pfleeger, 2002. “Ingeniería de Software, teoría y práctica”. Prentice Hall.

[McConnell, 1997], McConnell Steve. 1997. “Desarrollo y Gestión de Proyectos Informáticos”. McGraw Hill.

[Microsoft TechNet, 2006] “Metodología MSF aplicada”. [en red]. Disponible: http://www. microsoft.com/latam/technet/fases/msf.asp

[Miller y Lee, 2004], James Miller & Benegee lee, 2004. “Multi-Project Management In Software Engineering Using Simulation Modelling.”

[Miranda, 2002],Miranda. 2002 “Planning and Executing Time-Bound Projects”, IEEE Computer, pag Miranda, E., 73-79, Marzo 2002.

[Mohammad y García, 1996], Mohamamad R. Azarang, Eduardo García Dunna, 1996. “simulación y análisis de modelos estocásticos”. Mc Graw-Hill México.

[Montaño, 2003], Montaño García Agustín”,2003 “. Teoría de decisiones”. PAC.

[Montes, 1981], “Iniciación al método de Camino Critico” Agustín Montes G. 1981, editorial Trillas.

[Noori y Hamid, 1997], Noori y Hamid. 1997. “Administración de producción y operaciones” Mc. Graw-Hill

[Padberg, 2003], Frank Padberg. 2003 “Applying Process Simulation to Software Project Scheduling”. Prosim Workshop, Portland, USA.

[Perry y Wolf, 1992], Dewayne E. Perry, Alexander L. Wolf, 1992. “Foundations for the Study of Software Architecture”

[Pressman, 2002], Roger S. Pressman, 2002. “Ingeniería del Software Un enfoque practico”, Mc Graw Hill

[Ríos, *et al* 2000], David Ríos Insua, Sixto Ríos Insua, Jacinto Martín Jiménez, 2000. “Simulación Métodos y Aplicaciones”. Alfa omega, México.

[Rooco, 1967], Martino Rooco L. 1967. “Administración y Control de Proyectos” (Determinación de la Ruta Crítica). Editorial técnica S.A.

[Ropponen y Lyytinen, 2000], Ropponen and Lyytinen. 2000. “Components of Software Development Risk: How to Address them?,” IEEE Transactions on Software Engineering, Vol. 26, No. 2, Febrero de 2000.

[Standish Group, 1994], “The CHAOS Report (1994)”.[En Red], Disponible: http://www.standishgroup.com/sample_research/chaos_1994_1.php

[Yamal, 2002], Yamal Chamoun, 2002.“Administración Profesional de Proyectos” Mc Graw-Hill.

[Zorrilla, 1987], Santiago Zorrilla Arena, 1987. “Introducción a la Metodología de la Investigación. Ediciones Océano, S. A.

Anexo A. Algoritmo de un simulador basado en riesgos

Este algoritmo es una propuesta para crear un componente simulador, y contiene nueve pasos que son los siguientes:

1. **Inicializar las variables de entrada.** El usuario captura las actividades a realizar en el proyecto de software, los recursos que se asignan a cada actividad, el tiempo de duración, la disciplina a la que pertenece en el proyecto. Es necesario que el usuario visualice los riesgos en el proyecto por ejemplo: descripción, clasificación, causas por las cuales podría aparecer dicho riesgo, probabilidad de que pueda aparecer así como su impacto en el proyecto.

2. **Analizar los riesgos en el proyecto.** Una vez identificados y clasificados los riesgos, un componente de análisis del riesgo calculara la prioridad de los riesgos en base a su probabilidad e impacto en el proyecto.

3. **Generar la red de actividades.** La red de actividades en el proyecto inicia con la construcción de una red de actividades compuesta por varias actividades dependientes en algún punto del calendario. La red de actividades define la actividad que sigue y la anterior de una actividad.

4. **Mientras no finalice el proyecto iterar.** Mientras el proyecto no haya concluido se realiza una iteración para determinar la presencia de riesgos en el proyecto.

5. **Para toda actividad en un tiempo *t*, determinar si hay presencia de riesgo.** Cuando inicia la simulación se empieza a determinar si un riesgo aparecerá en el proyecto o no a través del tiempo, mediante una decisión aleatoria si un riego aparece se realiza un análisis de impacto del riesgo en todas las actividades del proyecto a partir del tiempo en que se generó el riesgo y se recalcula la red de actividades.

6. **Realizar un análisis de impacto para cada actividad.** Una vez aparecido un riesgo en el proyecto, se analiza cuál es el grado de afectación en cada actividad y los tiempos de retrasos de las mismas. Esto se efectúa mediante un componente que analiza todas las actividades relacionadas con el riesgo a partir del tiempo de aparición, después el componente calcula el grado de exposición e impacto que causará en la actividad y finalmente se determina si habrá o no un retraso en cada en la terminación de dicha actividad.

7. **Recalcular red de actividades.** La red de actividades se recalcula con la nueva información.

8. **Incrementar el tiempo**

   Se avanza en el proyecto con respecto al tiempo hasta que no haya ninguna actividad a analizar.

9. **Mostrar resultados de la simulación.** La información que se obtuvo de la simulación del proyecto es mostrada al administrador de proyectos por medio de una interfaz grafica.

1. Inicializar las variables de entrada
2. Analizar los riesgos en el proyecto
3. Generar la red de actividades
4. Mientras no finalice el proyecto iterear
5. Para toda actividad en un tiempo *t* determinar si hay presencia de riesgo
6. Realizar una análisis de impacto para cada actividad
7. Recalcular red de actividades
8. Incrementar el tiempo
9. Mostrar los resultados de la simulación