DISEÑO DE UNA ARQUITECTURA PARA UN ENTORNO DE MODELAMIENTO- SIMULACIÓN Y CREACION DE UN PROCESO PARA SU DESARROLLO POR UNA COMUNIDAD (I+D)

Ing. JORGE JAIR MORENO CHAUSTRE

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD INGENIERIAS FISICO-MECANICAS
ESCUELA DE INGENIERIA DE SISTEMAS E INFORMATICA
MAESTRIA EN INFORMATICA
BUCARAMANGA
2006

DISEÑO DE UNA ARQUITECTURA PARA UN ENTORNO DE MODELAMIENTO- SIMULACIÓN Y CREACION DE UN PROCESO PARA SU DESARROLLO POR UNA COMUNIDAD (I+D)

Ing. JORGE JAIR MORENO CHAUSTRE

Investigación presentada como requisito parcial para optar al título de
Magíster en Informática

Director del proyecto:

HUGO ANDRADE SOSA
Magíster en Informática

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD INGENIERIAS FISICO-MECANICAS
ESCUELA DE INGENIERIA DE SISTEMAS E INFORMATICA
MAESTRIA EN INFORMATICA
BUCARAMANGA
2006

DEDICATORIA

A Dios por no renunciar a mí a pesar de todo.
A mis Padres amados por esperar con paciencia.
A mi esposa Sandra por su amor, confidencia y apoyo.

AGRADECIMIENTOS

El desarrollo y conclusión exitosa de este proyecto se debe en gran parte a la colaboración desinteresada de muchas personas las cuáles me facilitaron enormemente la transformación completa de toda una idea de proyecto en un producto tangible que busca el beneficio de las comunidades de modelado y simulación a través de los servicios que ofrece la ingeniería del software. El autor desea expresar sus más sinceros agradecimientos a las siguientes personas y entidades:

Hugo Hernando Andrade Sosa, por toda la sabiduría y experiencia que me comunicó durante el desarrollo de esta tesis. Así mismo a todas las personas del grupo SIMON que de forma desinteresada, colaboraron con sus aportes, en especial al ingeniero Emiliano Lince desarrollador de Evolución 3.5.

A los miembros del grupo GTI del departamento de sistemas de la Universidad del Cauca, en especial a Miguel Niño, Carlos Cobos y Martha Mendoza por favorecer las condiciones necesarias para trabajar sin perturbaciones. También el agradecimiento va para Juan Carlos Vidal por sus valiosas orientaciones y a Erwin Meza por toda su colaboración desinteresada.

A la Universidad Industrial de Santander por favorecer las condiciones necesarias para culminar este proyecto, así mismo a la Universidad del Cauca por permitir el espacio para alcanzar este logro.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

ANEXO 14. RECURSOS DE REDACCION (ESPACIO EN BLANCO INTENCIONAL PARA CONSERVAR EL FORMATO DEL DOCUMENTO)

ANEXO 15. ENCUESTA COLCIENCIAS-SIMEP "PRÁCTICAS DE LA INGENIERÍA DEL SOFTWARE EN LAS PYMES DEL SUR-OCCIDENTE COLOMBIANO" (ESPACIO EN BLANCO INTENCIONAL PARA CONSERVAR EL FORMATO DEL DOCUMENTO)

ANEXO 16. FORMATO DE REGISTRO DE RESULTADOS DE LA EVALUACIÓN ARQUITECTÓNICA (ESPACIO EN BLANCO INTENCIONAL PARA CONSERVAR EL FORMATO DEL DOCUMENTO)

ANEXO 17. SUGERENCIAS A LA NUEVA ARQUITECTURA DEL ESMS-MI (ESPACIO EN BLANCO INTENCIONAL PARA CONSERVAR EL FORMATO DEL DOCUMENTO)

RESUMEN

TITULO: DISEÑO DE UNA ARQUITECTURA PARA UN ENTORNO DE MODELAMIENTO-SIMULACIÓN Y CREACION DE UN PROCESO PARA SU DESARROLLO POR UNA COMUNIDAD (I+D).*

AUTOR: JORGE JAIR MORENO CHAUSTRE **

PALABRAS CLAVE: Arquitecturas de Software, Entornos de Modelado y Simulación, Integración de Dinámica de Sistemas con otras Herramientas de Representación, Desarrollo en Comunidad.

DESCRIPCIÓN:

Este proyecto propone un diseño lógico para la arquitectura de un entorno software de modelado y simulación (ESMS) que permita extender la funcionalidad entregada por la herramienta EVOLUCION (desarrollada por SIMON), hacia la integración de componentes de modelado y simulación para otras herramientas de representación de conocimiento, desarrolladas por grupos de investigación (i+d) geográficamente dispersos, pertenecientes a la comunidad latinoamericana de modelado y simulación en dinámica de sistemas.

Además, con el propósito de apoyar el trabajo de los equipos (i+d) geográficamente dispersos cuando trabajan cooperativamente (desarrollando componentes para EVOLUCION), este proyecto propone una metodología de desarrollo de software en comunidad soportada mediante una plataforma de servicios en Internet, que integran herramientas adecuadas para la gestión y documentación de proyectos, así como el control, comunicación y coordinación de los equipos de desarrollo bajo un ambiente distribuido.

Finalmente, este trabajo, propone el diseño mismo de un proyecto comunitario de desarrollo de software con el propósito de implementar las características especificadas en la arquitectura lógica concebida para el Entorno Software de Modelado-Simulación, usando la metodología de software propuesta sobre la plataforma de servicios seleccionada para soportar la comunidad.

* Tesis de Maestría

** Facultad de Ingenierías Físico-Mecánicas. Programa: Maestría en Informática. Director: Mag. Hugo Andrade Sosa.

SUMMARY

TITLE: ARCHITECTURE FOR A MODELING-SIMULATION ENVIRONMENT AND CREATION OF A SOFTWARE PROCESS FOR ITS DEVELOPMENT BY A COMMUNITY.*

AUTHOR: JORGE JAIR MORENO CHAUSTRE**

KEYWORDS: Software Architecture, Modeling-Simulation Environments, Systems Dynamics and its integration with others representation tools, Developing Communities.

DESCRIPTION:

This project, proposes an architectural design for a modeling-simulation software environment (ESMS spanish initials) which allows to grow up EVOLUCION´s functionality (Developed by SIMON) in order to assure that modeling-simulation software components developed by others (R+D) groups can get plugged-in to EVOLUCION. Those modeling-simulation components were conceived to work with System Dynamics and others knowledge representation tools.

By the other hand, with the purpose to support the cooperative work of the (R+D) geographically dispersed groups (which will develop the EVOLUCION compatible components), this project proposes a communitarian developing software process supported by a services internet platform (called DotProject) which integrates software tools for manage projects and supporting coordination, control and coordination for developing teams under a distributed environment.

Finally, this work, proposes a communitarian software project to implement all the functional and non-functional characteristics specified by the logical software architecture conceived for the ESMS, using the proposed software methodology over the internet platform selected.

* Masters Thesis.

** Faculty of Physical and Mechanical Engineerings. Degree: Masters of Computer Science. Director: Mag. Hugo Andrade Sosa.

INTRODUCCIÓN

Este trabajo de grado se enmarca desde dos perspectivas afines pero que provienen de necesidades de innovación, desarrollo e investigación originadas en grupos de investigación diferentes: el GTI[4] de la Universidad del Cauca[18] y SIMON[14] de la Universidad Industrial de Santander. En consecuencia esta introducción se divide en dos partes: en primer lugar se describirá el interés de esta tesis de grado, desde la perspectiva de SIMON y en segundo lugar (no menos importante) el interés del mismo desde la perspectiva del GTI.

Interés de la tesis de grado desde la perspectiva de SIMON.

Durante los últimos ocho años el Grupo SIMON ha intentado llevar las ideas del Pensamiento Sistémico (P.S) a la educación y en particular, una expresión de este pensamiento, el Dinámico Sistémico, mediante la Dinámica de Sistemas (D.S). Estos esfuerzos se han dando a conocer en diversos eventos académicos. En el mismo sentido, el grupo SIMON, ha estado comprometido con el continuo aporte en materia de la investigación, desarrollo y soporte de herramientas software que apoyen el proceso de divulgación, aprendizaje y socialización de la dinámica de sistemas, el pensamiento sistémico y en general de los enfoques relacionados con el modelamiento y simulación de realidades complejas como soporte a la resolución de problemas, generación de conocimiento y toma de decisiones entre otros. Actualmente, SIMON, potencia mediante alianzas estratégicas con varios grupos de investigación (I+D) en D.S y otras áreas, pertenecientes a otras universidades colombianas, el desarrollo en comunidad de un Entorno Software de Modelamiento y Simulación de modelos integrados (ESMS), basado en la arquitectura software de sus herramientas EVOLUCION[1], HOMOS[5], entre otras. Para tal propósito SIMON está interesado en la definición e instrumentación de un proceso de desarrollo que lidie con los aspectos distribuidos de la comunidad en el marco de este macroproyecto.

Interés del trabajo de grado desde la perspectiva de GTI.

De otro lado, actualmente el grupo GTI[4]de la Universidad del Cauca[18], esta ejecutando el proyecto denominado SIMEP-SW[13] el cuál propende por:

- Generar un Sistema Integral de Mejoramiento de los procesos de desarrollo de software que proporcione las herramientas necesarias para motivar a las empresas a mejorar sus procesos de desarrollo de software con el objeto de facilitar el posicionamiento y la competitividad en mercados internacionales, y que se ajuste con el tiempo y su aplicación, a la industria del software en Colombia.

Los resultados esperados de SIMEP-SW[13] se enuncian a continuación:

- Un proceso ágil. Que guíe a un programa de mejora de procesos en el marco de un proyecto de mejora. Cuenta con los elementos básicos para hacer posible que una empresa pequeña o mediana que está surgiendo, pueda adelantar esfuerzos hacia la adecuación de un proceso de desarrollo acorde a sus necesidades.

- Un modelo de calidad liviano. Que integre proceso y producto. Que guíe la organización de las personas y los equipos, las disciplinas y las áreas de trabajo asociadas a la definición, aplicación y mejora del proceso hacia un nivel de madurez definido.

- Un modelo de Métricas del proceso productivo. Que permita medir el desempeño de los proyectos y que sea integrable a un modelo de costos de producción.

- Un modelo de Valoración del proceso productivo. Que permita valorar el cumplimiento con respecto a un modelo de Calidad (P.e. CMMI[12], Agile SPI Quality Model[13] , ISO[6], etc.)

- Un marco conceptual y tecnológico para soportar la tecnología de procesos (FrameWorkPDS). Consiste en una guía para la concepción y construcción de procesos. Una implementación basada en SPEM[15] para modelar, visualizar y aplicar procesos productivos.

- Y finalmente, un marco tecnológico para:

    - La definición, visualización y aplicación del FrameworkPDS.

    - La valoración– Quality Models's Evaluation Tool.

    - Soporte a los aspectos distribuidos del proceso. SEDISE (A Support Environment for Distributed Software Engineering).

    - Administración de proyectos de mejora– Project SPI Administrator Environment.

El interés de GTI[4] a través de SIMEP-SW, sobre este proyecto se concentra en el soporte a los aspectos distribuidos de los procesos de software en la industria, mediante la plataforma SEDISE[11].

En consecuencia y con el propósito de apoyar el compromiso de SIMON[14] de la UIS[19] al mismo tiempo de constituirse en un logro en la presentación de resultados del proyecto SIMEP-SW del grupo GTI de la Universidad del Cauca[18], este trabajo de grado pretende:

Para SIMON de la Universidad Industrial de Santander:

- Instanciar un proceso de desarrollo distribuido y soportarlo mediante una plataforma software de integración, cuyos servicios de valor agregado, apoyen, las actividades relacionadas con la gestión, comunicación, seguimiento y medición del proceso, con el propósito de favorecer las condiciones mínimas para apoyar a una comunidad de investigadores (I+D)

geográficamente dispersa en varias universidades del país, cuyo interés consiste en el desarrollo conjunto de ESMS de modelos integrados.

- Diseñar la línea base de la arquitectura lógica para un ESMS de modelos integrados, cuyo punto de partida, consiste en las arquitecturas de software de las herramientas desarrolladas por SIMON[14] (Evolucion[1] y Homos[5]). Este diseño, facilitará la implementación de la funcionalidad adicional del sistema, mediante la asignación de trabajo a equipos distribuidos de desarrollo.

Para el proyecto SIMEP-SW[13] GTI[4] de la Universidad del Cauca[18]:

- En el marco de SIMEP-SW extender el FrameworkPDS mediante capacidades de desarrollo distribuidas, constituyéndose en un avance hacia el futuro en la forma como las organizaciones de desarrollo de software en Colombia, enfrentarán el desarrollo en comunidad basado en comunidades virtuales de (I+D).

Para la Comunidad de (I+D) en Colombia:

- Propiciar las condiciones para favorecer una cultura de cooperación y colaboración entre investigadores geográficamente dispersos, estableciendo un precedente para futuros esfuerzos distribuidos de I+D, no sólo en el área del desarrollo de software orientado hacia el Modelamiento y Simulación, sino también en aquellas áreas, donde un proceso distribuido de desarrollo se considere como una opción seria para equipos geográficamente distantes o comunidades que emprenden un proyecto en común.

A continuación se describe brevemente el propósito de cada capítulo:

En el capítulo 1 se describe el planteamiento del problema que establece el objeto de estudio y la hipótesis de investigación que justifican el presente trabajo.

En el capítulo 2 se plantean los objetivos del trabajo que intentan satisfacer las necesidades esclarecidas a partir del problema.

En el capítulo 3 se construye el marco teórico que da soporte conceptual al trabajo propuesto y que está representado en el estudio de la evolución y conformación del desarrollo de software en comunidad, las arquitecturas del software, entornos de modelado-simulación y finalmente los modelos de mejoramiento para procesos de desarrollo. Todos ellos fundamentales para orquestar una solución apropiada a la problemática planteada para la investigación.

En el capítulo 4 se desarrolla el primer objetivo general del presente trabajo de tesis el cuál se propone: *Elaborar los requerimientos funcionales y diseñar la arquitectura base , para un ESMS de modelos integrados, cuyo punto de partida consiste en la arquitectura software de la herramienta* ***Evolución[1]*** *desarrollada por el grupo SIMON[14] de la escuela de ingeniería de sistemas de la UIS[19].*

En el capítulo 5 se desarrolla el segundo objetivo general del presente trabajo de tesis el cuál se propone: *Elaborar una instanciación de un proceso adecuado para el desarrollo de software en comunidad que favorezca las condiciones mínimas para unificar e integrar la comunicación y gestión durante la ejecución de un proyecto software orientado a entregar un "Entorno de Modelamiento-simulación de modelos integrados", liderado por el grupo SIMON.*

En el capítulo 6 se desarrolla el tercer objetivo general del presente trabajo de tesis, el cuál pretende: *Elaborar una propuesta ante COLCIENCIAS, que corresponda al diseño de un plan de proyecto (I+D) (de mediano plazo) para el ESMS de modelos integrados apoyado en el proceso distribuido y soportado en la arquitectura base diseñada, como alternativa de continuidad.*

Finalmente se expresan las conclusiones y recomendaciones derivaras de las experiencias relativas a la ejecución del presente trabajo de tesis.

1. PLANTEAMIENTO DEL PROBLEMA Y JUSTIFICACIÓN

De acuerdo con Marques (2004), en medio de un mercado extremadamente competitivo y dinámico, el desarrollo ágil de software de calidad, fiable y a bajo costo constituye una preocupación fundamental en la actual industria del software, la cuál según Pressman (2001) ha encontrado problemas que dificultan la creación de software con las características deseadas. Estos problemas (Pressman 2001) consisten en primer lugar, que el ritmo de la demanda de software supera al ritmo de la oferta del mismo, lo que significa que actualmente se crea software de una manera más lenta de lo que el mercado exige; en segundo lugar, la actual sociedad de la información ha fundamentado todas sus actividades en la confiabilidad de los sistemas informáticos, por lo cual una falla en el desempeño de estos sistemas podría causar sufrimiento humano o grandes perdidas económicas; y por último, muchos desarrolladores de software no poseen un pensamiento ingenieril y arquitectónico sólido implicando una concepción de arquitecturas software carentes de mantenibilidad, adaptabilidad, escalabilidad, portabilidad y robustez. Con el propósito de responder a estas preocupaciones, las organizaciones de software deben repensar constantemente sus paradigmas de desarrollo, al mismo tiempo que apropian procesos bien definidos, debidamente soportados con herramientas CASE y mejorados continuamente, de acuerdo con lo afirmado por Marques (2004). Es precisamente, durante este proceso de re-pensamiento de los paradigmas de desarrollo, que las organizaciones están convirtiendo en una norma, el desarrollo de software que involucra equipos geográficamente distribuidos, como respuesta a los complejos desafíos del mercado, de acuerdo al reporte del Yankee Group (2003) , “La compañía de producción monolítica con la capacidad de concebir, diseñar, hacer ingeniería, manufacturar y distribuir sus productos, es cosa del pasado, y así mismo aquella que asume la creación, prueba y distribución de nuevas aplicaciones propietarias desde un solo lugar”. Esta tendencia ha motivado la incorporación de aspectos colaborativos a la ingeniería y gestión de proyectos de desarrollo de software en ambientes distribuidos. El desarrollo distribuido aprovecha la sinergia

entre las personas pero corre todos los riesgos relacionados con la comunicación debido a la dispersión geográfica. En consecuencia, en un ambiente distribuido de desarrollo, es equivocado asumir por defecto, que los individuos pertenecientes a un equipo y hasta los mismos equipos, comparten la misma porción de espacio, tiempo, cultura y conocimiento, coordinando sincronizadamente sus respectivas actividades y decisiones. Por el contrario, un ambiente distribuido de desarrollo, en combinación con un mercado exigente y extremadamente dinámico, exige, que las organizaciones, se preocupen por orientar sus esfuerzos hacia:

- Invertir recursos, tiempo y dinero en la adecuación, modificación y mejoramiento de los procesos actuales, al mismo tiempo que apropian e implementan buenas prácticas de la ingeniería del software.

- Aprovechar las ventajas competitivas, derivadas del uso y aplicación de las innovaciones tecnológicas y herramientas CASE disponibles.

- Adquirir competencia y capacidad a la hora de gestionar equipos distribuidos de desarrollo, al mismo tiempo que potencian talentos y recursos sin importar su ubicación geográfica.

De otro lado, relativo al constante y creciente ritmo de la demanda de soluciones de software y en el marco de la academia-investigación-industria nacional, existen necesidades particulares que convocan a los profesionales del software a asumirlas con diligencia y responsabilidad con el propósito de convertir crisis en oportunidades. Y es precisamente en este sentido que una necesidad–demanda ha sido identificada relativa al desarrollo un ESMS de Modelos Integrados que compita en pertinencia, funcionalidad, calidad, fiabilidad y coste con otros ESMS propietarios disponibles en el mercado. Esta necesidad ha sido detectada por SIMON[14], quien desde hace tiempo ha impulsado proyectos (I+D) de software orientado al Modelamiento-Simulación en dinámica de sistemas y otros enfoques. Sin embargo, SIMON reconoce que atender en solitario tal necesidad, demandaría la puesta en

marcha de un proyecto, cuyo tamaño y esfuerzo implicarían un alto nivel de riesgo. SIMON y SIMEP-SW[13] del GTI[4], reconocen en el paradigma de desarrollo distribuido una alternativa interesante para superar el desafío de un proyecto de desarrollo para un ESMS de calidad industrial en el mediano plazo y la propuesta para una comunidad (I+D) geográficamente dispersa.

En este contexto, SIMON[14] ha asumido el liderazgo para un proyecto de desarrollo de un ESMS de modelos integrados, el cuál agrega a una red de investigadores pertenecientes a varias universidades colombianas. Este proyecto será desarrollado por equipos (I+D) heterogéneos y geográficamente dispersos. En consecuencia, SIMON, en su papel de líder, se verá obligado a invertir recursos y tiempo en mejorar la forma como el software es desarrollado y desplegado, requiriendo herramientas eficaces para administrar y unificar el conocimiento, la comunicación, la gestión y el personal (gestores, desarrolladores y clientes) mediante una plataforma que integre eficazmente todo el proceso de desarrollo en un ambiente distribuido. Además, SIMON requiere apoyo al momento de concebir una arquitectura base para el ESMS, sobre la cuál se conectarán unidades de implementación independientes, con interfaces bien definidas y desarrolladas por equipos geográficamente dispersos. Sin embargo, SIMON aún no cuenta con un proceso (bien definido y soportado informáticamente) adecuado para el desarrollo distribuido de software, puesto que la experiencia que cada universidad tiene se relaciona con desarrollo de software de manera centralizada. En consecuencia, si se empieza a trabajar sin mitigar las causas de los problemas arriba descritos, el proyecto de desarrollo, sufrirá entre otros los siguientes síntomas: dispersión, pérdida de control sobre la sincronización de las tareas y personas, fallas en la comunicación, falencias en la gestión, seguimiento y control del proceso. Por otro lado el grupo SIMEP-SW[13] del GTI[4]en un esfuerzo de ofrecer alternativas que mitiguen las debilidades y prevengan los riesgos de las prácticas del software nacionales (incluídas las de SIMON), ve con gran interés el desarrollo del presente proyecto, cuyo producto final constituiría un aporte significativo para ofrecer

alternativas de desarrollo que fomenten la cooperación, el tele trabajo y la comunicación entre redes de investigadores-desarrolladores geográficamente dispersas.

En consecuencia, el presente trabajo tiene como preguntas fundamentales de investigación las siguientes:

- **H1:** ¿Es posible diseñar proyectos de desarrollo para comunidades geográficamente dispersas de (I+D) cuyo propósito sea entregar soluciones software a la comunidad académico-investigativa e industrial que compitan en funcionalidad-costo-beneficio con software propietario disponible en el mercado?

- **H2:** ¿Crear las condiciones mínimas para favorecer la consolidación de una comunidad de desarrollo distribuido como estrategia para enfrentar problemas particulares de la academia, la investigación y/o la industria en el marco nacional, representa una alternativa seria al desarrollo tradicional de software propietario?

2. OBJETIVOS

2.1. OBJETIVO GENERAL

En el marco de un proyecto de desarrollo distribuido de software, ejecutado por una red de investigadores, geográficamente dispersa y heterogénea, este trabajo de tesis se propone lo siguiente:

1. Elaborar los requerimientos funcionales y diseñar la arquitectura base, para un ESMS de modelos integrados, cuyo punto de partida consiste en las arquitecturas software de las herramientas desarrolladas (Evolución[1] y Homos[5]) por el grupo SIMON[14] de la escuela de ingeniería de sistemas de la UIS[19].

2. Elaborar una instanciación de un proceso adecuado para el desarrollo distribuido de software que favorezca las condiciones mínimas para unificar e integrar la comunicación y gestión durante la ejecución de un proyecto software orientado a entregar un “Entorno de Modelamiento-simulación de modelos integrados”, liderado por el grupo SIMON.

3. Elaborar una propuesta ante COLCIENCIAS, que corresponda al diseño de un plan de proyecto (I+D) (de mediano plazo) para el ESMS de modelos integrados apoyado en el proceso distribuido y soportado en la arquitectura base diseñada, como alternativa de continuidad.

2.2. OBJETIVOS ESPECÍFICOS

- Los objetivos específicos se dividen en tres categorías, dependiendo de cuál objetivo general derivan. A continuación, se plantean los objetivos específicos relativos al objetivo general número uno (1).

- Elaborar la especificación formal de los requerimientos funcionales y de calidad para el entorno de modelamiento y simulación de modelos integrados que posteriormente será desarrollado distribuidamente por la comunidad (I+D).

- Elaborar una concepción lógica (estática y dinámica) para una arquitectura de software orientada a unidades de implementación independientes con interfaces bien definidas y asignables a equipos de desarrollo geográficamente distribuidos constituyentes de la comunidad (I+D).

- A continuación, se plantean los objetivos específicos relativos al objetivo general número dos (2).

  - Apoyar las responsabilidades del proceso, relacionadas con la gestión del ciclo de vida (fases, disciplinas e iteraciones), trabajadores (roles, responsabilidades y conformación de equipos de desarrollo) y documentación (modelos y artefactos), mediante una plataforma software.

  - Apoyar las actividades del proceso relacionadas con la comunicación y sociabilidad de los equipos de desarrollo distribuido (programación de reuniones, asignación, seguimiento y entrega de tareas, publicación de calendarios e informes del avance y estado del proyecto entre otros) mediante una plataforma software.

  - Evaluar y cuantificar el estado de un proyecto de desarrollo distribuido (que utilice el proceso), mediante el diseño de un modelo de métricas del proceso, con el propósito de soportar las responsabilidades relativas al monitoreo y control del proyecto.

  - Validar aspectos clave de mejora del proceso distribuido mediante la aplicación de un enfoque de SPI (Software Process Improvement) durante el transcurso de la ejecución del proyecto, con el propósito de entregar un proceso distribuido estable y con cierto nivel de madurez, listo para ser instanciado en el marco de una comunidad (I+D) geográficamente dispersa.

- A continuación, se plantean los objetivos específicos relativos al objetivo general número tres (3).

  - Elaborar un (1) documento para una propuesta de investigación ajustado al estándar de una convocatoria de COLCIENCIAS para el desarrollo de proyectos de I+D.

3. MARCO TEÓRICO

Los elementos fundamentales que guían el desarrollo de esta propuesta son:

La evolución del pensamiento relacionado con los procesos de desarrollo de software cuando se realizan bajo ambientes distribuidos geográficamente y como esta distribución afecta aspectos claves de la interacción humana, tales como : la gestión, la disponibilidad, la coordinación, la comunicación.

La preocupación constante de la industria del software por entregar a la sociedad de la información más y mejores soluciones de software de calidad a bajo costo, en un mercado exigente y extremadamente dinámico, mediante el constante cuestionamiento y perfección de las buenas prácticas y principios esenciales de la ingeniería del software.

La percepción del desarrollo centrado en las arquitecturas de software orientado a objetos y fundamentadas en interfaces bien definidas cuyo propósito es favorecer la sinergia en equipos de desarrollo independientes.

La fundamentación teórica relacionada con meta modelos (SPEM) de procesos de desarrollo de software y su especificación formal, permite un enfoque de instanciación sólido y disciplinado que garantiza la concreción, correctitud y efectiva comunicación y apropiación del proceso propuesto.

Una síntesis de cada uno de éstos se presenta a continuación.

3.1. PROCESOS DE DESARROLLO DISTRIBUIDO

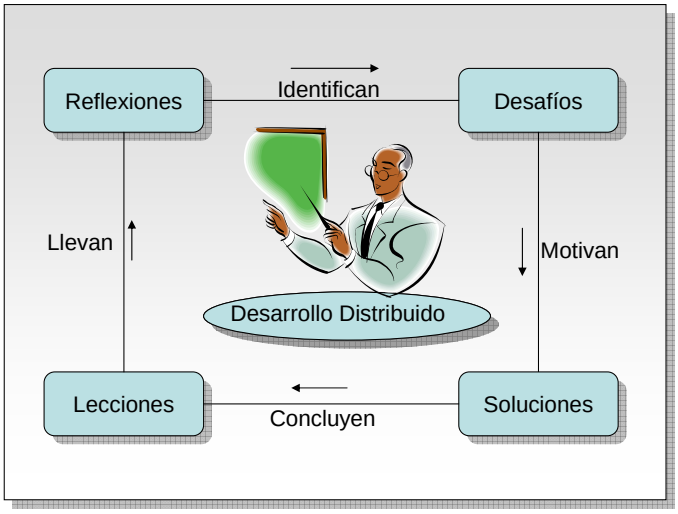

Figura 1: Modelo de Evolución del Pensamiento para procesos distribuidos

"El desarrollo distribuido establece desafíos significativos relativos a la cultura organizacional y la filosofía de la gestión, las arquitecturas de software, la comunicación, satisfacción de los clientes, los procedimientos de integración y prueba y muchas otras áreas." Dice Lockheed Martin Fred Palmer[16], quien se ha desempeñado como moderador del Open Forum Session durante el "The Executive Round Table. La evolución del concepto del desarrollo distribuido (ver Ilustración 1: Modelo de Evolución del Pensamiento para procesos distribuidos), tiene un ciclo de vida que inicia con las reflexiones suscitadas por la necesidad de superar con éxito los esfuerzos de desarrollo que no imponen restricciones respecto a la dispersión geográfica de todos los implicados. Estas reflexiones, se agrupan en tres categorías a saber: La gente, los Procesos y la Tecnología. A este respecto, se presenta a continuación, la revisión de algunos planteamientos y experiencias sobre los procesos de desarrollo distribuidos, basado en el SOFTWARE PRODUCTIVITY CONSORTIUM[16]:

- **La Gente.** La postura más relevante identificada para el desarrollo geográficamente distribuido es la comunicación. Dada la necesidad de crear una visión compartida del producto y proceso software en un ambiente de desarrollo distribuido, el valor de la comunicación cara a cara es vital, particularmente cuando se presenta un diseño complejo que represente una significativa dificultad técnica.

- **El Proceso.** La necesidad de procesos maduros en común a través de los proyectos de desarrollo geográficamente distribuidos es uno de los desafíos identificados más difíciles de superar. Sin embargo, el simple hecho de tener procesos en común no es suficiente, el verdadero reto de la gestión consiste en inculcar y hacer respetar rigurosamente esos procesos comunes entre todos los miembros del equipo y la gestión, y en asegurarse de que las herramientas seleccionadas para el proceso encajen todas en su lugar.

- **La Tecnología.** El groupware y las herramientas de comunicación son requisitos tecnológicos de apoyo a los procesos distribuidos de desarrollo. Así mismo, aplicaciones Intranet, servidores del proyecto, y video conferencia se han consolidado como las tecnologías más útiles en la mayoría de esfuerzos de desarrollo distribuido. Debe tenerse cuidado sin embargo en el uso de herramientas genéricas las cuáles frecuentemente están pobremente documentadas y soportadas y en ocasiones hacen el uso del proceso algo poco amigable. Asimismo, se destaca la importancia de negociar y acordar un conjunto de herramientas comunes de apoyo al proceso siempre que sea posible, aunque en ocasiones es más crítico aún el establecimiento de procesos comunes y arquitecturas abiertas para garantizar el éxito del desarrollo distribuido. Por último, la necesidad de compartir información privada en un entorno geográficamente distribuido, motiva la seguridad de la red distribuida.

De otra parte (continuando con la explicación de la Figura 1), mediante un continuo trabajo de la comunidad del software, las anteriores reflexiones esclarecen nuevos desafíos que traen consigo los ambientes distribuidos de desarrollo los cuáles fundamentalmente se concentran en formas alternativas para tratar los aspectos distribuidos de actividades clave tales como: La comunicación, la infraestructura, la coordinación, la disponibilidad y la gestión. El esclarecimiento de estos nuevos desafíos, provoca una reacción, cuyo propósito fundamental consiste en la concepción de soluciones que mitiguen los inconvenientes causados por las soluciones tradicionales al tratar con los aspectos distribuidos de las actividades clave arriba mencionadas. Sin embargo, estas soluciones son meras aproximaciones y son constantemente cuestionadas a la luz de su eficacia. En consecuencia, se pueden aprender muchas lecciones de su apropiación, uso y desempeño. A este respecto, se presenta la revisión de algunos de los paralelos Desafíos vs. Soluciones que implican los esfuerzos de desarrollo distribuido, así como las lecciones aprendidas de la implementación e implantación de las mismas en entornos distribuidos. Estos aportes fundamentalmente pertenecen a Michael Kircher, Prashant Jain, Angelo Corsaro y David Levine, quienes lideran la iniciativa del Dispersed Extreme Programming o DXP (Kircher 2001). De otra parte, se exploran las diferencias entre el desarrollo distribuido y el desarrollo disperso, destacadas por Scott W. Ambler (Ambler 2002), con el objeto de encuadrar el universo temático objetivo de este trabajo de investigación. La información relativa a los desafíos y soluciones inherentes a los procesos distribuidos de software se encuentra descrita a continuación:

3.1.1. Desarrollo Distribuido: Desafíos encontrados y soluciones propuestas.

Desafío: La Comunicación. Un aspecto importante en la comunicación para los humanos, es conocer como reaccionan las personas a las cosas que uno mismo dice. Para juzgar esa reacción, típicamente una persona podría leer el lenguaje

corporal o facial y hasta la entonación de la voz. En el desarrollo distribuido las personas generalmente se encuentran en ubicaciones geográficamente distantes, ¿como podría alguien conocer las reacciones que otros tienen acerca de sus comentarios? A continuación, se plantean varias estrategias para superar el componente distribuido de la comunicación.

- Iniciar la creación del equipo de trabajo en una ubicación física única, con el propósito de permitir a las personas el tiempo suficiente para conocerse y constituir una cultura en común. Luego, es suficiente con una pequeña ventana de video, para percibir, las mutuas reacciones que en la comunicación remota tienen las otras personas acerca de los comentarios de otros. Al respecto podrían acordarse formas de comunicación remota basadas en video, teléfono, Chat, email o tele conferencia.

- Acordar reuniones periódicas para incrementar las relaciones interpersonales de todos los miembros del equipo, de tal manera que la cooperación remota se facilite gracias a los lazos de confianza creados.

- Finalmente la capacidad y habilidad para compartir documentos mejora enormemente la comunicación y cooperación remotas.

Desafío: La Coordinación. Cuando dos o más miembros del equipo trabajan juntos en un proyecto desde ubicaciones físicas diferentes, la coordinación de su trabajo, se convierte en todo un desafío. Sincronizar la disponibilidad ajustando las diferencias en los tiempos y cronogramas personales, con el propósito de coordinar la distribución, integrar las actividades, y al mismo tiempo compartir documentos y aplicaciones es un desafío a superar. A continuación, se plantean varias estrategias para superar el componente distribuido de la coordinación.

- La apropiada coordinación entre los miembros del equipo requiere un mínimo de planeación. Sin embargo hacer un uso extensivo de varias líneas de comunicación puede facilitarlo. Por ejemplo: 2 miembros del proyecto

ubicados remotamente podrían intercambiar sus agendas de actividades diarias vía e-mail, en ellas podrían asignar de común acuerdo algunas franjas para dedicarlas al proyecto. Lo anterior les obligaría a tomar en cuenta las diferencias de tiempo disponible existentes para cada uno.

Desafío: La Infraestructura. Tanto la comunicación como la coordinación, en el desarrollo distribuido dependen de la infraestructura disponible. Esto incluye tanto el hardware como el software, así como el ancho de banda de la red. Una infraestructura pobre puede dificultar seriamente estos aspectos en un proceso distribuido. La disponibilidad de una infraestructura suficiente es vital. A continuación, se plantean varias estrategias para superar el componente distribuido de la infraestructura.

- Todos los miembros del equipo deben tener el software y hardware adecuado. Se constituyen como criterios importantes de selección del software los siguientes: El software debe ser seleccionado, teniendo en cuenta su: Facilidad de uso, interoperabilidad con otras herramientas y disponibilidad en diferentes plataformas. Igualmente, el hardware debe ser seleccionado de forma cuidadosa. De hecho cada desarrollador necesitará el poder de una estación de trabajo clásica con características multimedia.

Desafío: La Disponibilidad. Los miembros de un equipo distribuido de desarrollo pueden estar disponibles pero no necesariamente de forma síncrona. Algunos de ellos incluso podrían estar trabajando en múltiples proyectos y tener su dedicación restringida, mientras que otros simplemente tendrán una disponibilidad limitada debido a asuntos de índole personal. En otras ocasiones podrían presentarse incluso limitantes de disponibilidad relacionadas con diferentes zonas horarias. A continuación, se plantean varias estrategias para superar el componente distribuido de la disponibilidad.

- No negar ayuda a nadie que la esté solicitando especialmente si es desde una ubicación remota.

- Publicar diaria o semanalmente las agendas de disponibilidad para cada miembro del equipo.

Desafío: La Gestión. El gestor del equipo, necesita confiar mucho en sus subordinados y más aún si ellos frecuentemente son remotos. Deben definirse nuevas estrategias alternativas al control directo que realiza el gestor. A continuación, se plantean varias estrategias para superar el componente distribuido de la gestión.

- Los líderes del proyecto necesitan aprender a manejar equipos distribuidos. En particular, debe aprenderse como administrar a los miembros ubicados en locaciones remotas esto podría incluir: Requerir diaria o semanalmente reportes de todos los miembros del equipo ya sean locales o remotos.

- Proporcionar realimentación frecuente a todos los miembros del equipo para crearles un sentimiento de conexión y pertenencia. Eventos regulares pueden ayudar a construir confianza y motivación entre todos los miembros del equipo.

Finalmente (continuando con la explicación de la Ilustración 1: Modelo de Evolución del Pensamiento para procesos distribuidos), nuevas soluciones llevan a nuevas lecciones y a partir de estas, nuevas reflexiones son planteadas, no para finalizar el ciclo, sino para iniciar uno nuevo, incrementando el conocimiento y estabilizando el concepto siempre evolutivo del desarrollo distribuido.

3.1.2. Desarrollo Distribuido Vs. Desarrollo Disperso

Según Ambler (2002), en el desarrollo distribuido, los sub-equipos trabajan desde varias ubicaciones geográficamente remotas, pero se comportan como un solo gran

equipo. Este tipo de desarrollo ocurre, por ejemplo, cuando, un equipo externo construye parte del sistema mediante Out-Sourcing, mientras que otros equipos trabajan juntos dentro de la misma organización. Por el contrario, el desarrollo disperso, ocurre cuando:

- Cada miembro del equipo trabaja en un lugar remoto, por ejemplo, su propia casa.
- Los individuos trabajan en ubicaciones físicas diferentes dentro de las instalaciones de una misma compañía.
- Los individuos trabajan bajo el esquema de código abierto.

En síntesis, los dos enfoques difieren radicalmente, mientras que en el desarrollo distribuido, son los sub-equipos quienes trabajan desde diferentes ubicaciones remotas, en el desarrollo disperso son los individuos quienes trabajan remotamente. El interés de este trabajo de investigación se concentra en el desarrollo distribuido, debido a que es la configuración organizativa que mapea fielmente la realidad de distribución de la red de investigadores de I+D en modelamiento y simulación que está involucrada activamente en el proyecto de desarrollo liderado por SIMON[14] y la oportunidad de proponer un logro para el proyecto SIMEP-SW[13] del GTI[4] perteneciente a la Universidad del Cauca.

3.1.3. Antecedentes sobre Procesos de Desarrollo Distribuido

En la actualidad se pueden encontrar trabajos de investigación y desarrollo con contenidos similares al tema que se quiere desarrollar en este proyecto (I+D), los cuales sirven como referentes para el mismo. A continuación se mencionan algunos de estos trabajos realizados y se resaltan las similitudes, diferencias y reflexiones respecto del trabajo de grado que se quiere desarrollar:

Investigación en Procesos Pesados en entornos distribuidos: Modelo de Administración de Proyectos: propuesta para la ejecución en un entorno de desarrollo de software físicamente distribuido (Zanony 2002).

- **Descripción.** Propone un modelo de administración de proyectos que incluya el UP y usando UML, para el desarrollo de software de e-business, en un entorno físicamente distribuido. El modelo esta dividido en seis fases: definición de requerimientos, exploración y definición del proyecto, producción de procesos, evaluación, transición e integración. En el futuro, la intención es desarrollar un soporte software para el modelo y aplicar este software dentro del entorno en estudio.

- **Resultado Obtenido.** El modelo propuesto difiere de los presentes en que busca la unión entre el desarrollo de procesos en entornos distribuidos y la administración de procesos, incorporando un ciclo de vida en espiral, la orientación a objetos y el lenguaje UML. Un énfasis especial se ha dado en la etapa de Definición de Requerimientos y la integración de fases, como los resultados mas afectados en el entorno de desarrollo distribuido de software. El modelo propuesto fue el resultado de la necesidad de encontrar respuestas a un problema crítico que estaba presente en los entornos de trabajo. Este problema esta centrado en las dificultades de comunicación debido a la distancia física y cultural entre grupos de usuarios y desarrolladores. Debido al poco tiempo no fue posible aplicar el modelo propuesto.

- **Reflexión.** Esta investigación presenta similitudes con el trabajo que se quiere realizar. Por lo tanto los problemas que se encontraron en este antecedente son casi los mismos que se enfrentarán para el desarrollo de este proyecto. Sin embargo esta investigación solo busca el desarrollo de software de e-business mientras que el presente trabajo de grado busca la concepción de una metodología para desarrollo de software en general. Una

carencia de la investigación anterior es que no se puso en práctica por no contar con el tiempo necesario. Se espera que este trabajo facilite el soporte informático para el proceso distribuido propuesto, con el objeto de beneficiar a una comunidad (I+D) geográficamente dispersa en varias universidades del país interesadas en el emprendimiento de proyectos de desarrollo de entornos de modelamiento y simulación de modelos integrados. De otra parte, los resultados de este trabajo, constituirán un logro de valor agregado al proyecto SIMEP-SW[13] de la Universidad del Cauca.

Aplicación de un proceso ágil en entornos distribuidos: Distributed eXtreme Programming (DXP) (Kircher 2001)

- **Descripción.** XP enfatiza la necesidad de tener los miembros del equipo de trabajo físicamente cercanos. Sin embargo, por varias razones, esto no siempre puede ser posible. Para mejorar esta situación, un grupo de personas ha propuesto una idea llamada “Distributed eXtreme Programming” (DXP), la cual involucra los méritos de XP y lo aplica en un entorno de un equipo distribuido. La experiencia obtenida por este grupo de trabajo muestra que DXP puede ser efectiva y que merece la pena en proyectos con equipos que están distribuidos geográficamente. DXP ofrece muchos desafíos, los cuales pueden superarse. DXP consigue afectar cuatro prácticas de XP, para las cuales se proponen las siguientes soluciones:

    - **Planificar.** Cuando existe dispersión geográfica entre los desarrolladores y los clientes, el proceso de planificación puede apoyarse mediante video conferencia y las aplicaciones compartidas remotamente.

    - **Programación en parejas.** La Programación en Parejas Remota (RPP) debe ser usada, cuando los miembros del equipo no se encuentran en la misma ubicación física. Esto requiere

videoconferencia y soporte para aplicaciones compartidas, que compartan el Entorno de Desarrollo Integrado (IDE).

- **Integración Continua.** Si un miembro del equipo esta trabajando en un sitio central, el/ella puede invitar a otro miembro remoto para realizar integración en la máquina. Si ambos miembros son remotos, esto no es posible.

- **Cliente in situ.** La videoconferencia deberá ser usada para involucrar al cliente remoto. El cliente necesita cumplir un cierto conjunto de reglas tales como coordinación y disponibilidad.

- **Resultados Obtenidos.** El Proyecto: Software llamado "Web-Desktop" proporcionará el entorno de trabajo para DXP. Este software es accesible vía Web y permitiría a un miembro del equipo usar cualquier PC conectado a la Internet y tener la misma sensación de estar trabajando en su propio entorno de trabajo. Por otra parte, un miembro del equipo de trabajo móvil podría entrar a un café Internet, usar una cámara Web y/o su micrófono y conseguir conectarse al resto de sus compañeros de equipo. No seria necesario descargar e instalar software en cada máquina que el miembro usa.

- **Experiencia Vivida.**

  - **Planificar.** Se ejecutaron varias sesiones de videoconferencia con el cliente. Se uso un editor regular y una aplicación compartida de software.

  - **Programación en Parejas.** Se asignaron historias de usuarios a las parejas y se comenzó el proceso de desarrollo. Se utilizo videoconferencia y aplicaciones compartidas. Se utilizo el e-mail para asignar citas según el horario.

- **Integración Continua.** Se integraron los cambios desde el lugar de desarrollo hasta el lugar principal.

- **Cliente en el sitio**. Se utilizo videoconferencia, también el e-mail para comunicar la hora y el canal para las sesiones de videoconferencia.

- **Reflexiones.** Usando una combinación de comunicación síncrona, como la videoconferencia, y comunicación asíncrona, como el e-mail, resulta ser más efectivo. Sin embargo, lo que se pierde es la presencia física lo cual nunca puede ser completamente sustituida con una herramienta de videoconferencia. El desarrollo paralelo lanza el problema de la integridad del código fuente. Para eso se utilizo un e-mail token para serializar el acceso a cambios a los equipos trabajando en una sección de código común. Este proyecto ofrece una buena fuente de información debido a que la investigación realizada en cuanto a DXP fue colocada en práctica. Sin embargo es solo una extensión de XP para procesos de desarrollo distribuido por lo que presenta mayor aplicación para procesos de desarrollo pequeños. Lamentablemente la herramienta software desarrollada en esta investigación no está disponible para el público en general, ya que podría ser un gran punto de apoyo para la herramienta que dará soporte a este proyecto.

Antecedente de un entorno de soporte para un proceso de desarrollo distribuido: PROYECTO GÉNESIS [3]

- **Descripción**. Propone el desarrollo de un entorno distribuido que soporta la ingeniería del software cooperativa, permitiendo mantener coordinadas y controladas las actividades relativas al el desarrollo de software aunque se realicen desde ubicaciones geográficamente distantes o simplemente ***UD´s***. Cada ***UD*** podría ejecutar instancias de un proceso (o sub-proceso) de

desarrollo, las entradas y/o salidas de esos procesos (o sub-procesos) son en realidad artefactos de software (ejemplo: módulos ejecutables, documentos de diseño, pruebas etc.). Cada ***UD*** mantiene la autonomía de sus procesos internos y sus tecnologías de soporte (por ejemplo: sistemas de gestión de artefactos y sistemas de gestión del proceso). La definición de la organización distribuida del proceso de desarrollo GENESIS propone la necesidad de distinguir entre dos niveles:

- **El primer nivel**: consiste en la definición del proceso global (también conocido como el proyecto), el cual describe el flujo de trabajo del proceso de desarrollo de software, los artefactos elaborados y como los sub-procesos son asignados a las diferentes ***UD´s***.

- **El segundo nivel:** permite a una única ***UD***, refinar el subproceso asignado a esta, mediante una definición del proceso local compuesto por actividades. Con el propósito de definir tanto el proceso local como el global, un lenguaje de definición de procesos debe ser provisto, con instrucciones específicas para construir y gestionar las tareas (y sus actividades) a nivel local y global.

- **Reflexión.** Una de las propuestas de GENESIS[3], es la capacidad de gestionar los aspectos dinámicos del proceso de software, específicamente las desviaciones que pueden ocurrir en cualquier momento durante la ejecución del proyecto. Este proyecto es semejante al que se plantea en el presente proyecto, con la diferencia que génesis esta liderado por una comunidad de empresas europeas, por lo que su alcance y objetivos son mas ambiciosos, mientras que el actual trabajo plantea una solución al paradigma de desarrollo distribuido adecuado para una comunidad (I+D) geográficamente dispersa en varias universidades del país, que participan en desarrollos conjuntos de entornos de simulación y constituye un logro para el proyecto SIMEP-SW[13] de la Universidad del Cauca[18]. Sin embargo en el

portal de este proyecto[3] solo muestran algunos resultados hasta 2002, no pudiéndose determinar el éxito y/o culminación del mismo.

3.2. ARQUITECTURA DE SOFTWARE

En este apartado, se desea explorar el concepto de arquitectura de software y su importancia, las estructuras de arquitecturas software más comunes y su propósito, los atributos de calidad de una arquitectura, el ciclo de vida de la arquitectura y el proceso de desarrollo centrado en la arquitectura.

3.2.1. Definición de Arquitectura de Software.

"La arquitectura de software de un programa o sistema de cómputo, consiste en la estructura o estructuras del sistema, las cuales comprenden aquellos elementos de software, sus propiedades externas y las interrelaciones entre ellos." (Bass 2003). A continuación se descompone la anterior definición con el objeto de proporcionar más claridad sobre el concepto de arquitectura.

- **Las propiedades externamente visibles.** Para un elemento de software, estas se corresponden con todos aquellos servicios, características de desempeño, manejo de errores, uso de recursos compartidos, que otros elementos de software asumen de cualquier otro elemento.

- **La arquitectura define elementos de software.** La arquitectura encapsula la información acerca de cómo los elementos de software se relacionan con otros, omitiendo aquella información que no representa interacción. De esta forma, la arquitectura es algo así como una abstracción del sistema que suprime aquellos detalles de los elementos de software que no tienen que ver con información relacionada con como estos usan o son usados y se relacionan o interactúan con otros. En los sistemas modernos, los elementos de software interactúan con otros por medio de interfaces bien definidas, las cuales dividen los detalles de implementación de un elemento

en dos partes, la pública y la privada. La descripción de la arquitectura software hace énfasis en mostrar los aspectos públicos de los elementos de software que la constituyen.

- **La arquitectura comprende una o mas estructuras.** Todos los proyectos con cierta complejidad, son divididos en unidades de implementación independientes, esas unidades satisfacen ciertas responsabilidades, las cuáles frecuentemente constituyen la base de la asignación de trabajo para los equipos de desarrollo. Este tipo de unidades de implementación, constituyen aquellos datos y programas de carácter público que pueden ser invocados desde otras unidades de implementación así como aquellos programas y datos que son privados. Este tipo de estructura es frecuentemente utilizado para describir el sistema, sin embargo, es bastante estática, ya que se enfoca fundamentalmente en como se subdivide la funcionalidad del sistema en unidades de implementación que son asignadas a equipos de desarrollo. Otros tipos de estructuras, por el contrario, se enfocan más en la forma en la que los elementos de software interactúan con otros en tiempo de ejecución con el propósito de satisfacer la funcionalidad del sistema.

- **Todo sistema de software tiene una arquitectura.** Esta afirmación tiene sentido, dado que todo sistema puede mostrarse como una colección de elementos y sus interrelaciones. Sin embargo existe una diferencia fundamental entre la arquitectura y su representación. Es decir, la arquitectura puede existir independiente de su documentación. Por ejemplo: tenemos un sistema software en ejecución (código binario), pero la documentación de su arquitectura se ha perdido o nunca se produjo.

- **El comportamiento de un elemento es parte integral de la arquitectura.** Tal comportamiento es aquel que permite a los elementos interactuar con otros, el cuál es ciertamente parte esencial de la arquitectura.

3.2.2. Importancia de la Arquitectura en el Software.

La arquitectura es importante por las siguientes razones:

- **Comunicación entre los Stakeholders**. La arquitectura de software representa una abstracción común del sistema que se utiliza como base para el entendimiento, negociación y acuerdos generales entre todos grupos implicados en el proyecto.

- **Decisiones de diseño tempranas.** La arquitectura de software se constituye como un manifiesto de las decisiones de diseño tempranas que se han tomado acerca de la estructura de un sistema. Esas decisiones tendrán implicaciones importantes a la hora de desarrollar, desplegar o mantener el sistema software.

- **La arquitectura es una abstracción re-utilizable entre sistemas.** La arquitectura de software constituye un modelo relativamente pequeño e intelectualmente manejable acerca de cómo se encuentra estructurado un sistema y como sus elementos trabajan juntos. Este modelo puede utilizarse en otros, sistemas, manteniendo similares atributos de calidad y requerimientos funcionales al mismo tiempo que favorece la reutilización a gran escala entre sistemas.

3.2.3. Estructuras de una arquitectura de Software y sus vistas.

El neurólogo, ortopedista, hematólogo y el dermatólogo todos tienen una vista diferente de la estructura del cuerpo humano. Todos se concentran en subsistemas particulares de este. Aunque esas vistas son todas diferentes y tienen cada una propiedades específicas, están relacionadas entre si, y juntas describen la anatomía del cuerpo humano. De la misma forma, el software actual es demasiado complejo como para tratar con todos sus aspectos de una sola vez. En lugar de eso, es preferible restringir la atención en cada momento a una de las estructuras (o un

pequeño conjunto de estructuras) de software. En consecuencia, para comunicar información significativa acerca de la arquitectura, debe tenerse muy en cuenta sobre cuál o cuáles estructuras se están abordando en un momento determinado. Los términos “estructura y vista”, guardan una estrecha relación cuando se discute la representación de una arquitectura. A continuación se definen cada uno de ellos:

- **Definición de Vista.** Una vista es un conjunto coherente de elementos arquitecturales y sus interrelaciones, realizado por y para los Stakeholders.

- **Definición de Estructura.** Es el conjunto de elementos mismo tal y como ellos existen en el hardware o el software. Las estructuras arquitecturales pueden ser divididas en tres grandes grupos dependiendo de la naturaleza misma de los elementos que intentan mostrarse.

    - **Estructura basada en Módulos.** Aquí los elementos son módulos, los cuáles se constituyen como unidades de implementación. Los módulos son una forma de representar al sistema basada en el código y sus áreas de responsabilidad. Este tipo de estructura, nos permite responder a preguntas tales como: Cuál es la función primaria o responsabilidad asignada a cada módulo? Que otros elementos de software podrían utilizar a un módulo? Que otros elementos de software podría utilizar este módulo? Como se relacionan los módulos (herencia, agregación, composición) entre sí?

    - **Estructuras de Conectores y Componentes.** Aquí los elementos son los componentes en tiempo de ejecución (unidades principales de computación) y los conectores (vehículos de comunicación entre componentes). Este tipo de estructuras permite responder a preguntas tales como: Cuales son los componentes principales en ejecución y como interactúan? Cual es la porción principal de información que se comparte? como evoluciona esa información a

través del sistema en ejecución? Que partes del sistema pueden correr en paralelo? Como cambia la estructura del sistema a medida que este se ejecuta?

- **Estructuras basadas en la Ubicación de los elementos.** Este tipo de estructuras muestra la relación entre los elementos de software y los elementos del entorno donde este es creado y ejecutado. Este tipo de estructuras permite responder a preguntas tales como: Que procesador ejecuta que elemento de software? En que ficheros es un elemento de software guardado durante el desarrollo o la prueba? Como es la asignación de los elementos de software a los equipos de desarrollo?

3.2.4. Cualidades de una Arquitectura Software.

Para describir las cualidades de una arquitectura de software, existen dos (2) grandes grupos de atributos a saber:

- **Cualidades Funcionales.** La funcionalidad consiste en la capacidad que tiene una arquitectura de software para realizar el trabajo correcto para el cual ha sido diseñada. El sistema entregará un desempeño deseable, siempre y cuando, las responsabilidades hayan sido correctamente asignadas a los elementos de software. Las cualidades funcionales son evidentes a los clientes del sistema, dado que entregan valor agregado cuando se ejecutan.

- **Cualidades No Funcionales.** Las cualidades No-Funcionales únicamente son evidentes a los clientes del sistema cuando fallan o están ausentes. En este grupo de cualidades se encuentran:

- **Cualidades del Sistema.** En esta categoría se encuentran la disponibilidad, modificabilidad, desempeño, seguridad, facilidad de prueba y usabilidad.

- **Cualidades del Negocio.** En esta categoría se encuentran Time-to-Market, Costo/Beneficio, Tiempo de vida esperado del sistema, Mercado Objetivo, Integración con otros sistemas predominantes.

- **Cualidades Implícitas.** En esta categoría se encuentran la integridad conceptual, la correctitud, la completitud y la facilidad de construcción.

3.2.5. Un Proceso Centrado en la Arquitectura: El Proceso Unificado

Existen muchos modelos de procesos que asumen diversas posturas para enfrentar el problema del desarrollo software. Sin embargo una postura se destaca entre las demás por su espíritu de desarrollo centrado en la arquitectura, planteando iteraciones cortas y re-alimentación frecuente con el propósito de agregar funcionalidad parcial del sistema con cada iteración, entregando versiones limitadas una vez se hayan implementado un conjunto significativo de características. Este proceso es denominado “Proceso Unificado de Desarrollo de Software”. El proceso unificado, planteado por (Booch 2000) es un marco de trabajo genérico que puede especializarse para diferentes sistemas de software, áreas de aplicación, niveles de aptitud y diferentes tamaños de proyectos. Está basado en componentes, lo cual implica que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas. Según Jacobson, Booch y Rumbaugh (Booch 2000) “el Proceso Unificado se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental”. A continuación se explican estas tres características:

- **Dirigido por casos de uso.** Los casos de uso representan los requisitos funcionales y todos juntos constituyen el modelo de casos de uso, el cual

describe la funcionalidad total del sistema. El proceso de desarrollo software, se dirige por los casos de uso según el riesgo que estos representan desde el punto de vista de la arquitectura, es decir, se realizan en primer lugar aquellos casos de uso que son arquitecturalmente críticos y luego los demás.

- **Centrado en la arquitectura.** El concepto de arquitectura incluye los aspectos estáticos y dinámicos más significativos del sistema. En síntesis la arquitectura comprende la estructura de cómo los componentes de software (lógicos o físicos) se organizan, interrelacionan y colaboran para satisfacer los requisitos críticos del sistema. La arquitectura surge de las necesidades de la empresa, como las perciben los usuarios y los inversores y se refleja en los casos de uso. Sin embargo, también se ve influida por otros factores tales como: la plataforma en la que tiene que funcionar el software, los bloques de construcción reutilizables de que se dispone, consideraciones de implantación, sistemas heredados y requisitos no funcionales. La arquitectura es una vista del diseño completo con las características más importantes resaltadas, dejando los detalles de lado. Al implementar los casos de uso más críticos desde el punto de vista de la arquitectura, el proceso unificado busca que se estabilice una línea base para esta, de tal forma que más tarde se agregue simplemente la funcionalidad restante. Los casos de uso y la arquitectura se relacionan porque cada producto tiene tanto función como forma. Ninguna es suficiente por sí misma. En esta situación, la función corresponde a los casos de uso y la forma a la arquitectura; debe haber una interacción entre ellas. Tanto la arquitectura como los casos de uso deben evolucionar en paralelo

- **Iterativo e Incremental**. El proceso unificado de desarrollo es *“adaptativo”*, esto quiere decir que divide un proyecto de software en partes más pequeñas o mini-proyectos. Cada mini-proyecto es una iteración que resulta en un incremento de software que realiza uno o más requisitos funcionales

del sistema. Las iteraciones hacen referencia a pasos en el flujo de trabajo y los incrementos al crecimiento del producto software. Para una efectividad máxima, las iteraciones deben estar controladas: esto es, deben seleccionarse y ejecutarse de una forma planificada.

3.2.6. El Ciclo de Vida de la Arquitectura de Software

Una de las cuestiones fundamentales relativa a las arquitecturas de software consiste en hallar la relación entre la arquitectura de software de un sistema y su entorno en tiempo de diseño y ejecución. Una primera aproximación, a esta cuestión, consiste en la percepción inmediata de que son los requerimientos funcionales, quienes determinan la forma de la arquitectura. Sin embargo esta percepción, podría mantener en la oscuridad otros factores esenciales que también determinan la forma de una arquitectura de software. Estos factores a su vez son influenciados por la arquitectura, al mismo tiempo que guían y regulan el trabajo del arquitecto de software. Las arquitecturas son influenciadas por los Stakeholders, la organización que desarrolla, la experiencia de los arquitectos y el entorno tecnológico. En síntesis, la arquitectura de software se constituye como el resultado de un ciclo de influencias de índole social, empresarial y técnica. Su existencia misma influye en los entornos sociales, empresariales y técnicos que subsecuentemente influirán a las arquitecturas futuras. Esto es mejor conocido como ciclo de influencias, desde el entorno hacia la arquitectura y de vuelta al entorno o El Ciclo de Negocio de la Arquitectura (Architecture Business Cycle – ABC) registrado por Bass (2003). Ningún enfoque de desarrollo centrado en la arquitectura podría ignorar este fenómeno. En consecuencia, los intentos de desarrollo centrados en la arquitectura deben asumir los desafíos correspondientes a la elaboración de la descripción del negocio, entendimiento de los requerimientos, selección, comunicación, evaluación, implementación y aseguramiento de la calidad de la arquitectura del software.

Las actividades involucradas en la creación de una arquitectura de software, la realización del diseño basado en la arquitectura, su implementación y la gestión que permite reorientar constantemente todo el proceso hacia la consecución de un sistema software deseado, se enuncian a continuación:

- **Elaborar una investigación del mercado para el nuevo sistema.** Investigar la necesidad del mercado para un nuevo sistema, implica averiguar respuestas para preguntas tales como: Cuanto debería costar el desarrollo del producto, cuál es el tiempo adecuado que tenemos para lanzarlo al mercado? Necesitará conectarse con otros sistemas? Cuáles son las limitaciones técnicas, sociales o legales del sistema? En todas estas cuestiones deben estar involucrados los arquitectos de software.

- **Entender los requerimientos.** Las técnicas de elicitación de requerimientos buscan encontrar aquellas cualidades funcionales y no funcionales que darán forma a la arquitectura y que dependen de las influencias sociales, tecnológicas y del entorno.

- **Crear o seleccionar una arquitectura.** Las arquitecturas son creadas o seleccionadas por su capacidad para satisfacer los requerimientos esclarecidos durante la elicitación.

- **Comunicar la arquitectura.** Para que la arquitectura sea verdaderamente efectiva como columna vertebral del diseño del sistema, esta debe ser comunicada de forma clara y no ambigua a todos los Stakeholders. Los desarrolladores deben entender las asignaciones de trabajo requeridas, los ingenieros de pruebas deben entender su papel, los gestores, deben comprender las implicaciones de la agenda de trabajo etc. Para alcanzar este objetivo, la documentación de la arquitectura del sistema debe ser clara, no ambigua, correcta, entendible por todos y actualizada constantemente.

- **Evaluar la Arquitectura**. En cualquier proceso de diseño, existen muchos candidatos a ser considerados. Algunos serán rechazados inmediatamente, mientras que otros disputarán el primer lugar. Escoger racionalmente entre todos los posibles diseños, es uno de los desafíos del arquitecto. Evaluar la arquitectura es esencial para asegurar que el sistema en construcción satisface todas las expectativas de los Stakeholders del proyecto. Algunos enfoques utilizados para evaluar una arquitectura de software son: Architecture Tradeoff Analysis Method (ATAM) (Kazman 1998).

- **Implementar la Arquitectura.** Esta actividad se preocupa de mantener el trabajo de los desarrolladores, fiel a las estructuras e interacciones reguladas por la arquitectura. Tener una descripción de arquitectura explícita, bien comunicada, es clave para asegurar la conformidad arquitectural. Igualmente disponer de un ambiente apropiado de apoyo al equipo, ayuda a crear y mantener una arquitectura correcta.

- **Asegurar la conformidad de la Arquitectura.** Finalmente cuando una arquitectura es creada y usada, esta va a la fase de mantenimiento. Se requiere una vigilancia constante para asegurar que la representación actual es fiel a la arquitectura en todo momento.

3.3. AGILE SOFTWARE PROCESS IMPROVEMENT (SPI).

SPI permite a las organizaciones mejorar significativamente la calidad del software que desarrollan, al mismo tiempo que su productividad y desempeño. El desafío para las pequeñas y medianas organizaciones (PYMES) en la actual industria del software consiste en: cómo aplicar **SPI** a la luz de sus objetivos organizacionales y como entregar sus productos de forma rápida al mismo tiempo que se implementa software iterativa e incrementalmente. El mejoramiento del proceso de desarrollo, involucra la aplicación de enfoques de mejora bien definidos, sistemáticos y repetibles al desarrollo de software. Para las PYMES, la aplicación de un enfoque

de **SPI** se constituye en un obstáculo a superar cuando: su implementación requiere un esfuerzo considerable en coste, escala y dedicación al mismo tiempo que los beneficios tangibles solo se perciben al cabo de varios años. En ese caso, lo ideal es la aplicación de un paradigma de **SPI** liviano, que sea efectivo, cuyo esfuerzo de inversión sea pequeño, que presente resultados tangibles en tiempos razonables y crezca en armonía con la disponibilidad de recursos.

Un paradigma de **SPI** liviano, debería estar en armonía con la cultura del desarrollo a pequeña escala, por ejemplo: pequeños cronogramas, fechas límites a corto plazo, proyectos dinámicos y presupuestos limitados. Un **SPI** Liviano debería concordar con principios propuestos por otros paradigmas que han demostrado ser exitosos y ampliamente utilizados. El problema para las PYMES consiste en como aprovechar, todo el conocimiento adquirido, la experiencia ganada y la tecnología desarrollada para impactar positivamente el mejoramiento de sus procesos de desarrollo en sus realidades específicas. Cuatro (4) requerimientos fundamentales para un **SPI** liviano, emergen de la necesidad de satisfacer las preocupaciones arriba planteadas:

- **Requisito 1.** Ser efectivo y producir buenos resultados (por ejemplo: ciclos de desarrollo más cortos, detección de errores más temprana y buen retorno de la inversión).

- **Requisito 2.** Ser incremental en términos de que debería ser factible de implementarse inicialmente con poco esfuerzo y luego ser extendido incrementalmente mediante ciclos cortos que coincidan con los ciclos de desarrollo de los proyectos a los cuales se apliquen.

- **Requisito 3.** Proveer resultados rápidos y tangibles de tal forma que pueda justificarse su continuación.

- **Requisito 4.** Utilización de tecnologías existentes ya probadas en este campo.

IMPACT como paradigma liviano para **SPI** propuesto por (Scout et. al 2002): Un "Framework" cíclico compuesto por los estados: entender, mejorar, aplicar y medir. Pueden aplicarse incrementalmente a lo largo de varios proyectos. En cada ciclo de aplicación, un enfoque diferente relativo al mejoramiento es puesto en marcha. De otra parte, al implementar iteraciones sucesivas las metas de mejoramiento serán alcanzadas rápidamente (usualmente dentro de unos pocos meses) y son factibles de medirse a través de los proyectos en los cuales el enfoque de mejoramiento elegido ha sido aplicado. El framework de IMPACT ha sido implementado utilizando herramientas existentes y tecnologías probadas

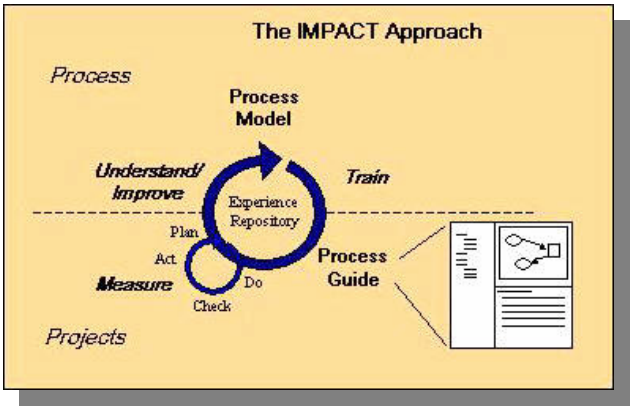

Figura 2: Nivel de Proyecto y Proceso en IMPACT

A continuación se describe brevemente cada nivel del enfoque SPI de IMPACT:

Ciclo del Proyecto. IMPACT proporciona un marco adecuado para administrar y ejecutar proyectos dentro de un **SPI** al nivel de proyecto. Aquí, muchos proyectos son realizados aplicando buenas prácticas relacionadas con la gestión de proyectos. Aquí puede resultar útil la aplicación de un enfoque de calidad típico como: "Planear, Hacer, Evaluar y Actuar". Este enfoque está dirigido por objetivos y está constituido por los siguientes seis pasos:

1. Descripción del proyecto y su entorno.

2. Definición de Objetivos (realistas, alcanzables y medibles).

3. Definición de los modelos de proceso y herramientas de soporte.

4. Ejecución del proceso, construcción de los productos, recolección y validación de datos y sus correspondientes análisis con el propósito de establecer re-alimentación y acciones correctivas.

5. Evaluación de las Prácticas actuales, definición de problemas, registro de errores y determinación de recomendaciones para futuros esfuerzos de mejoramiento.

6. Empaquetamiento de la experiencia ganada en modelos de conocimiento con el fin de proveer un acceso rápido a soluciones.

Ciclo del Proceso

El ciclo del proceso provee un marco para administrar el ***SPI*** mismo a un nivel organizacional. A continuación se enuncian las etapas fundamentales del ciclo del proceso:

1. Entender el proceso actual.

2. Mejorar el proceso actual

3. Elaborar la Guía del Proceso Mejorado.

4. Valorar el impacto de las mejoras.

Los beneficios esclarecidos durante la aplicación de un caso práctico en la empresa Allette Systems - Australia (Scott 2002) son los siguientes:

- La concentración de actividad a través de plantillas ha contribuido al éxito global del esfuerzo de SPI. La documentación resultante de cada proyecto guarda un nivel satisfactorio de consistencia.
- Mejora en la estimación y gestión del proyecto.
- Mejora en el entendimiento del trabajo y las relaciones con los clientes.
- Mejora en la percepción de los clientes relativa a la formalización del proceso y documentación. Esto ha sido visto como un enfoque más ordenado y profesional, aunque algunos clientes han tenido preocupaciones acerca del coste de producción de la documentación y el retardo en la escritura de código.
- Manejo de plantillas por parte de clientes y desarrolladores mejorando la comunicación y entendimiento del proceso.
- Percepción del efecto de las mejoras en 3 meses.

3.4. COMUNIDADES VIRTUALES

Desde el nacimiento hasta la muerte, damos forma y somos formados por las comunidades a las que pertenecemos. Para bien o para mal esas comunidades influencian nuestro lenguaje, como pasamos el tiempo, lo que consideramos importante, con quienes interactuamos y que naturaleza tiene esa interacción. Algunas comunidades crean condiciones favorables para interacciones sociales fuertes, mientras que otras son todo lo contrario, algunas son constructivas y algunas no. Desarrollar una comunidad virtual es una actividad compleja, puesto que el concepto de las comunidades virtuales tiene diversos significados para diferentes personas (Preece 2003). De otra parte, la revolución de la información y el Internet han mudado la forma en que las organizaciones e individuos se

relacionan. Este fenómeno es causado por aspectos como la globalización, los avances tecnológicos y la agresiva competencia industrial. Un nuevo tipo de organización descentralizada, geográficamente dispersa y con un propósito centrado es la ***"Comunidad Virtual"***, la cuál consiste en una colección de diversas entidades distribuidas geográfica, funcional o culturalmente, enlazadas mediante las TIC´s, estableciendo relaciones laterales y dinámicas para su coordinación, todo esto para satisfacer un propósito bien definido (DeSanctis 1998). A pesar de la naturaleza difusa de este tipo de organización, una identidad común impera en la mente de todos sus miembros, clientes y participantes, dentro de una compañía sin paredes (Galbraith 1995) que se comporta como una red de personas que colaboran entre si, trabajando juntas para la satisfacción del propósito fundamental de la comunidad. Una comunidad virtual esta caracterizada por los siguientes elementos fundamentales:

- **Las personas.** Interactúan socialmente para satisfacer sus propias necesidades o las de otros, desempeñando diversos roles en la comunidad.

- **Un Propósito en común.** El interés, la necesidad, el intercambio de información o servicios de valor agregado que dan sentido a la existencia y continuidad de la comunidad.

- **Las Políticas.** Los rituales, protocolos, reglas y leyes que guían la interacción y regulan el comportamiento entre los miembros de la comunidad.

- **Los Sistemas de Computadoras.** Soportan, facilitan y median la interacción social y el sentido de pertenencia a la comunidad, mediante el establecimiento de lazos electrónicos.

- **Limites permeables.** Una entidad puede pertenecer a varias organizaciones, las fronteras físicas no existen.

- **Una estructura re-configurable.** Se pueden redefinir procesos, roles, asignaciones y carga de trabajo según las necesidades.

En cualquier organización, la comunicación es considerada como factor fundamental de éxito. Sin la comunicación vía electrónica las comunidades virtuales simplemente no podrían ser y menos aún tratar con los inconvenientes causados por la distribución geográfica, temporal y cultural. Este tipo de comunicación, no solo permite a los miembros de una comunidad mantener relaciones virtuales con contactos ya establecidos, sino que provee una plataforma para fomentar nuevas relaciones de este tipo. En otras palabras, el verdadero poder de una comunidad virtual, ocurre cuando, las relaciones entre gente conectada electrónicamente producen cuantitativamente nuevos y diversos tipos de comunicación que permite hacer innovación en productos, servicios y procesos. Finalmente, la concepción de comunidad virtual como organización se relaciona estrechamente con cuatro conceptos según (OSOY Omar): groupware, equipos virtuales, tele-trabajo y oficina virtual. Esta tesis, se concentrará en las preocupaciones derivadas del funcionamiento eficiente y eficaz del equipo virtual. Los equipos virtuales se pueden comunicar sincrónicamente o asincrónicamente por medio de tecnologías como el e-mail, grupos de discusión, conferencias de audio/video/data, votación electrónica y el trabajo colaborativo. Algunos de los beneficios de los equipos virtuales soportados en herramientas de comunicación electrónicas, es que facilitan compartir información y conocimiento sin el costo que implican los desplazamientos físicos.

3.5. ENTORNOS DE MODELADO Y SIMULACIÓN

3.5.1. Exploración del concepto de Modelo

Un modelo es una herramienta conceptual para observar un sistema complejo desde un punto de vista particular. Muchos modelos son necesarios para contrarrestar los efectos de ceguera causados por uno solo. La metodología de modelamiento multi-faceta propuesta por Zeigler (1986) consiste en un enfoque

concebido para el modelamiento simulación que reconoce la irreducible complejidad de la realidad al mismo tiempo que afirma que modelos parciales pueden ser útiles cuando se construyen para la toma de decisiones limitadas. Zeigler (1986) afirma que una base organizada de modelos puede integrar cada perspectiva parcial para alcanzar un objetivo macro coherente. Las perspectivas de modelamiento se caracterizan por tomar en cuenta cada uno de los siguientes aspectos:

- Un único objetivo para el modelo.
- Un nivel simple de agregación
- Un único nivel a lo largo del eje de estructura – comportamiento.
- Un único formalismo de modelamiento o un paradigma específico de programación.

3.5.2. El Entorno de Modelamiento y Simulación

El concepto de un entorno de modelamiento y simulación, combina varios aspectos del proceso relacionados con el desarrollo y simulación de los modelos en una poderosa y completa colección de herramientas integrada. Conceptualmente un entorno de modelamiento y simulación armoniza el lenguaje de simulación con las herramientas de soporte (Haigh 1994). Otra forma de concebir un entorno de modelamiento y simulación consiste en describirlo como aquellas partes del sistema que el usuario percibe (incluye el sentimiento psicológico del sistema basado en las bondades de su funcionalidad) más allá de la superficial interfaz de usuario hasta las herramientas de soporte al proceso de modelamiento y simulación realizado por el usuario. Más específicamente, un ambiente es la integración sinergética de módulos de software para proveer un esquema fuerte y cohesivo cuyo propósito consiste en formular y resolver un conjunto dado de tareas (Arthur 1994).

3.5.3. Requerimientos para un Entorno de Modelamiento y Simulación

El mundo real del modelamiento y simulación, es una amalgama de diferentes entornos tratando de alcanzar los mismos objetivos. Aunque esos entornos proveen características similares ellos raramente utilizan el mismo lenguaje de modelamiento y más aún, obstáculos tales como el aprendizaje de un nuevo paradigma, modelos y formatos de datos incompatibles no permiten la explotación del trabajo previo mediante la reutilización de componentes ni el intercambio de modelos entre entornos. En consecuencia, definir requerimientos para un entorno de modelamiento y simulación debe ser un proceso que:

- Brinde soporte a las necesidades del usuario final, al mismo tiempo que minimiza la necesidad de aprender un nuevo lenguaje, mediante la especificación de características funcionales y no funcionales útiles

- Determine la correcta selección de la plataforma hardware/software y los lenguajes de desarrollo utilizados.

- Brinde soporte a la simultánea ejecución e interacción de múltiples modelos, permitiendo que modelos implementados en distintos lenguajes se integren en otros más grandes.(Haigh 1994)

- Soporte la portabilidad de un modelo de simulación desde un sistema a otro, permitiendo que un modelo desarrollado por una persona (analista, docente, consultor o estudiante) u organización pueda ser re-utilizado por otras personas u organizaciones en otros sistemas (Schwetman 1994).

- Brinde Soporte al modelador en la adopción de un enfoque de múltiples perspectivas para todos los aspectos mencionados en el numeral 3.5.1 (Zeigler 1986).

Los siguientes tres capítulos, describen la solución encontrada para los objetivos generales del presente trabajo de tesis.

4. ARQUITECTURA SOFTWARE PARA UN ESMS-MI[1]

Este capitulo se concentra en la satisfacción del segundo objetivo general de este trabajo, el cual propone:

- *"Elaborar los requerimientos funcionales y diseñar la arquitectura base, para un ESMS de modelos integrados, cuyo punto de partida consiste en las arquitecturas software de las herramientas desarrolladas (Evolución[1] y Homos[5]) por el grupo SIMON[14] de la escuela de ingeniería de sistemas de la UIS[19]."*

Según Clements (2003), la *arquitectura de software*, incluye en una colección de vistas los componentes principales del sistema, su comportamiento e interacciones, describiendo la forma en que estos colaboran para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta del sistema que filtra los detalles, con el propósito de comunicar las decisiones más importantes de diseño.

Una noción más amplia sobre la *arquitectura de software* consiste en la expresión de la organización fundamental de un sistema encarnada en sus componentes fundamentales, las relaciones entre ellos y con su ambiente y los principios que orientan su diseño y evolución. La arquitectura de software puede considerarse como el "puente" entre los requerimientos del sistema y la implementación, estructurando los aspectos funcionales críticos y relevantes del sistema como una colección de subsistemas interrelacionados y especificados en diferentes vistas que proveerán el soporte a la funcionalidad final deseada por el usuario y por el equipo de desarrollo. La arquitectura de software es el artefacto del ciclo de vida más temprano que refleja las decisiones de diseño más significativas y se considera como parte integral del proceso de desarrollo. Es importante destacar que muchos

[1] Entorno Software de Modelamiento y Simulación para modelos integrados.

de los atributos de calidad del software (Eficiencia, Fiabilidad, Portabilidad, Robustez) entre otros - son de naturaleza arquitectónica.

En consecuencia, es fundamental valorar la calidad de la arquitectura con el propósito de afirmar la propia calidad del software. En este aspecto, gracias a los *métodos de evaluación arquitectónica* es posible hacer predicciones fiables acerca de los atributos de calidad arquitectónica, basadas en técnicas más o menos formales (escenarios, modelos matemáticos, simulación, etc.).

Posteriormente a la aprobación del plan para este proyecto, se determinó que toda la arquitectura del ESMS-MI tendría como punto de partida sólo la herramienta EVOLUCION 3.5, debido a que su concepción arquitectónica es la más madura para realizar un tratamiento sobre la escalabilidad hacia un entorno mas sofisticado para el modelado y la simulación. Por tanto, para la concepción de la nueva arquitectura, se consideró fundamental e indispensable, la realización de una evaluación arquitectónica (Ver Numeral 4.1) que mediante la aplicación de un modelo de métricas sobre EVOLUCION 3.5, permita conocer su estructura interna, el estado de sus atributos de calidad, los posibles problemas de diseño subyacentes y finalmente las directrices para su eventual mejora, temas que se abordan a continuación.

4.1. METODOLOGIA DE EVALUACION ARQUITECTONICA

La metodología de evaluación arquitectónica aplicada a EVOLUCION 3.5 comprendió diez (10) pasos que abarcan desde el planteamiento de los objetivos mismos de la evaluación, hasta el nuevo diseño arquitectónico de EVOLUCION 3.5 cuya arquitectura abierta al desarrollo en comunidad le permitirá convertirse en el nuevo **ESMS-MI**. En la Figura 3, se ilustran los diez (10) pasos que guiaron la evaluación arquitectónica de EVOLUCION 3.5. Estas etapas se describen a continuación en la misma secuencia que se muestra en la Figura 3.

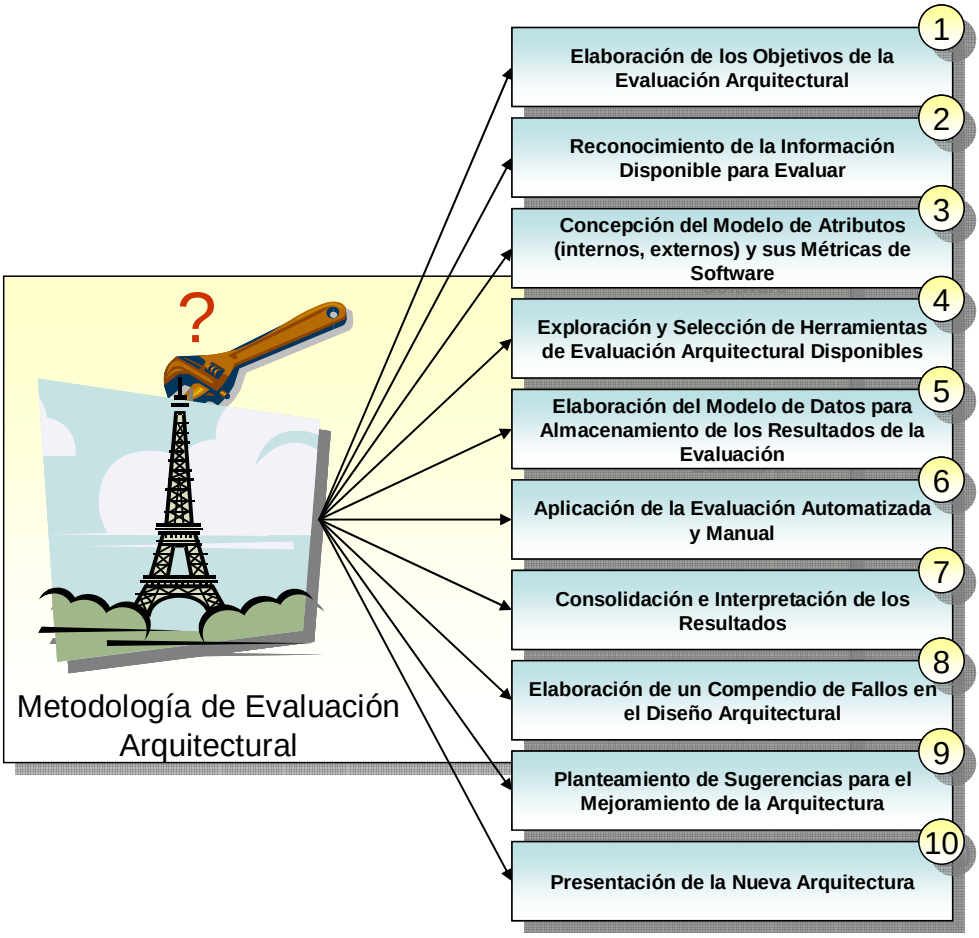

Figura 3: Metodología Propuesta para Evaluación Arquitectónica

4.1.1. Objetivos de Evaluación Arquitectónica

- Verificar el estado de conformidad de la descripción arquitectónica actual de EVOLUCION 3.5 y la arquitectura ya implementada en el código fuente entregado por SIMON.

- Valorar el estado de la arquitectura implementada con el propósito de encontrar fallos de diseño y sobre esta base proponer mejoras orientadas hacia la apertura de la misma en el marco de una comunidad de desarrollo.

- Documentar la Arquitectura implementada de EVOLUCION 3.5 integrando las mejoras sugeridas en la medida de lo posible.

4.1.2. Reconocimiento de la Información Disponible

La información disponible para llevar a cabo la evaluación arquitectónica de la herramienta EVOLUCION 3.5 se obtuvo mediante la colaboración del Ing. Emiliano Lince[2]. La fuente de información disponible se distribuye en tres categorías: requisitos, diagramas y código fuente. El código fuente pertenece al lenguaje de alto nivel DELPHI 7.0 y la versión de UML utilizada en los diagramas es la 2.0. A continuación en la Tabla 1, se enumeran los artefactos disponibles para la evaluación arquitectónica de EVOLUCION 3.5.

Categoría	Información Disponible
Requisitos	Especificaciones de Casos de Uso: **75**
	Número de Diagramas de Casos de Uso: **8**
Diagramas	Número de Diagramas de Clase: **4**
	Número de Diagramas de Secuencia: **5**
	Número de Diagramas de Componentes: **1**
	Número de Diagramas de Paquetes: **1**
Código	Número de Clases: **448**
	Número de Paquetes: **239**
	Líneas de Código: **59.277**

Tabla 1: Información Disponible para Evaluar Evolución 3.5

4.1.3. Concepción del Modelo de Atributos y Métricas

En este apartado, se explora de forma breve el concepto de calidad arquitectónica y sus atributos (externos, internos) asociados. Luego a partir de varios autores, se intenta concebir un modelo de métricas que valoren el estado de algunos atributos arquitectónicos con el propósito de clarificar lo que será evaluado en EVOLUCION 3.5.

---

[2] Miembro del Grupo SIMON y desarrollador de la versión 3.5 de EVOLUCION

La calidad del software significa a menudo cosas diferentes para diferentes personas. Por ejemplo: **Para los usuarios finales -** Un sistema tiene calidad si soporta efectivamente sus tareas de forma ágil y confiable. **Para el Administrador del Sistema -** Un sistema tiene calidad, si es fácil de instalar y adaptar a nuevas configuraciones de hardware sin causar efectos colaterales sobre otro software ya instalado. Finalmente, **para los desarrolladores.** El sistema tendrá calidad si puede adaptarse fácilmente a nuevos requerimientos.

De otra parte, los atributos o características conforman la noción de calidad de un producto de software y pueden ser externos o internos. Los atributos externos, son "***visibles***"externamente (de ahí su nombre) por ejemplo: la *confiabilidad* y la *mantenibilidad*, pueden ser medidas en términos de cómo el software se relaciona con su entorno y sólo cuando el producto ha sido creado. En cambio, un atributo interno puede ser medido en términos del producto mismo, por ejemplo el *tamaño*, *acoplamiento* y la *cohesión*, pueden determinarse a partir de su representación UML o su código fuente. De esta forma, los atributos internos, pueden medirse durante y después de la creación del producto, sin embargo, estos atributos no son evidentes durante la ejecución del software y no tienen significado en si mismos, sin embargo, tienen un impacto directo sobre los atributos de calidad externos. Por ejemplo, un *alto acoplamiento* o una *baja cohesión* implican una alta complejidad estructural, la cuál, a su vez, tiene un impacto directo sobre la *facilidad de mantenimiento* y *de prueba*.

En los siguientes dos (2) numerales se exponen brevemente los atributos de calidad externos e internos para valorar la arquitectura de software implementada en EVOLUCION 3.5.

4.1.3.1. Arquitectura y sus Atributos de Calidad Externos

En la Figura 4, se ilustran las cualidades esenciales externas que un buen diseño arquitectónico debe reflejar según Pressman (2001), las cuáles se definen brevemente a continuación:

- **La conformidad funcional:** Una buena arquitectura de calidad debería implementar todos los requisitos explícitos contenidos en el modelo de análisis y debe acomodar todos los requisitos implícitos que desea el cliente.

- **Adaptabilidad:** Consiste en el nivel de esfuerzo requerido para realizar cambios en una arquitectura.

- **Modularidad:** Esta cualidad se concentra en promover el principio del ocultamiento de la información.

- **Entendible:** De acuerdo con Somerville (1998), el entendimiento estará afectado por: La cohesión, el acoplamiento, la nominación, la documentación y la complejidad.

    - **Cohesión:** Es una consecuencia del ocultamiento de la información. La meta es hacer que los componentes sean lo más cohesivos posible, asegurándose de que sus responsabilidades estén altamente relacionadas entre si, ofreciendo el mayor nivel de funcionalidad posible, afirma Pfleeger (1998).

    - **Acoplamiento:** De acuerdo con Pressman (2001), este es un indicador de la fuerza de interconexión entre los componentes o elementos de la arquitectura. Los sistemas altamente acoplados tienen una fuerte interconexión, lo que se refleja en una gran dependencia entre sus componentes internos y una alta sensibilidad

al mantenimiento. Mantener el acoplamiento en un nivel bajo es saludable para la robustez de la arquitectura, afirma Pfleeger (1998).

La Figura 4, muestra los atributos externos de la arquitectura que pueden medirse con el propósito de valorar su calidad y en la Tabla 2 se describen estos mismos atributos en relación con sus atributos subordinados. En consecuencia, la información que se especifica a continuación, se constituye en el modelo de métricas externo concebido para valorar EVOLUCION 3.5.

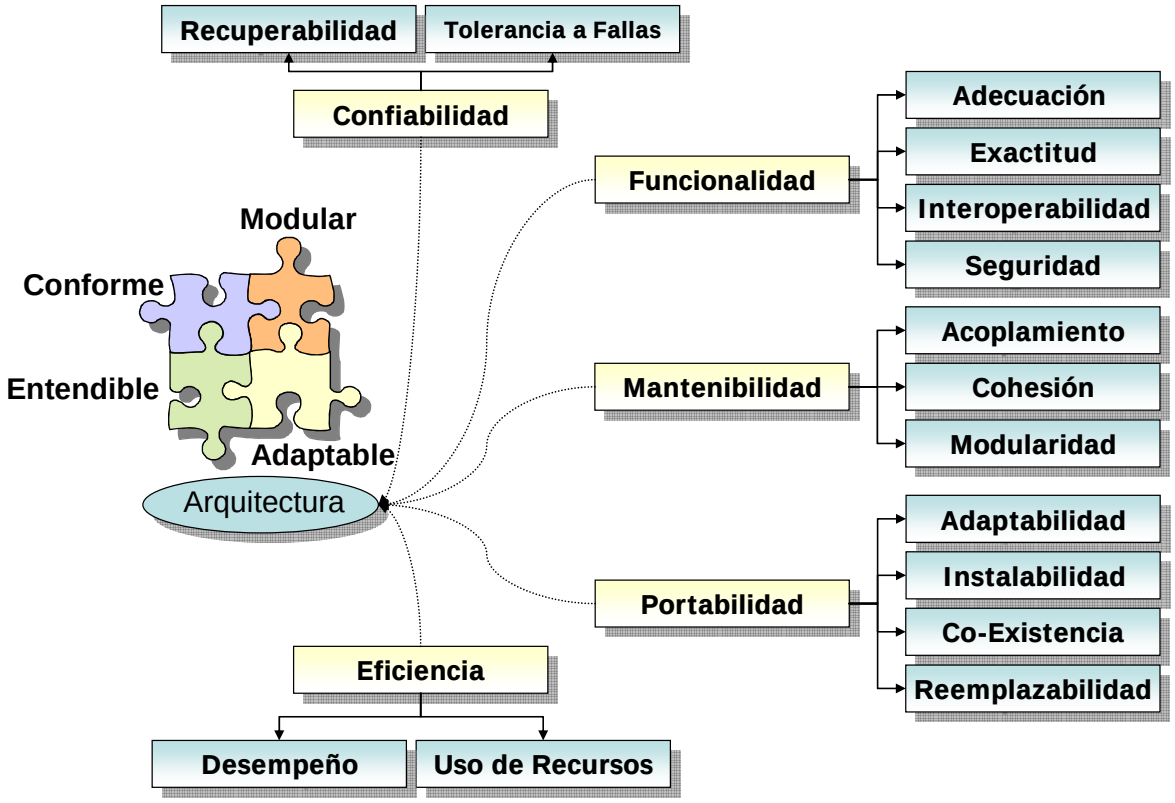

Figura 4: Atributos de Calidad de la Arquitectura

Atributo de Calidad	Definición	Sub-Atributo	Definición
Funcionalidad	Según Kazman (2001) consiste en la Habilidad de un sistema para realizar el trabajo para el cual fue concebido.	**Adecuación/Corrección**	Grado en el que un programa satisface las especificaciones y cumple los objetivos del usuario.
		Exactitud / Fiabilidad	Grado en el que un programa se espera que realice su función con una precisión requerida. Por ejemplo: entrega de cálculos numéricos confiables.
		Ínter operación	Capacidad de interacción con sistemas externos.
		Seguridad/ Integridad	Capacidad de brindar soporte al control de acceso a la información privada.
Confiabilidad	De acuerdo con Barbacci (1995), esta consiste en la medida de la habilidad que tiene un sistema para mantenerse operativo a lo largo del tiempo.	**Tolerancia a Fallas**	Capacidad para manejar excepciones
		Recuperabilidad	Existencia de mecanismos o dispositivos de software para reestablecer el nivel de desempeño y recuperar datos.
Eficiencia	De acuerdo con Bass (2003), esta se refiere a la cantidad de comunicación e interacción existente entre los componentes del sistema.	**Desempeño**	Entrega de un rendimiento deseable durante la ejecución.
		Uso de Recursos	Utilización eficiente de los recursos aún en condiciones de escasez.
Mantenibilidad	Según Bosch (1999), esta consiste en la Capacidad de modificar el sistema de manera rápida y a bajo costo.	**Acoplamiento**	Medida de la dependencia e interacción entre componentes.
		Modularidad	Numero de componentes que dependen de otro.
Portabilidad	De acuerdo con Pressman (2001), esta consiste en la habilidad del sistema para ser ejecutado en	**Adaptabilidad**	Capacidad para adaptarse durante su ejecución a una configuración cambiante de recursos.

Atributo de Calidad	Definición	Sub-Atributo	Definición
	diferentes ambientes de computación.	**Instalabilidad**	Facilidad de instalación en diversos ambientes de computación.
		Coexistencia	Facilidad para operar adecuadamente sin sufrir o causar perturbaciones de otro software ya instalado.
		Reemplazabilidad	Lista de componentes reemplazables para cada Componente.

Tabla 2: Descripción de los Atributos Externos de Calidad

4.1.3.2. Arquitectura y sus Atributos de Calidad Internos

Una vez determinado el modelo de atributos externos, el siguiente paso consiste en concebir y armonizar una colección de métricas internas que valoren el estado de la información disponible (véase numeral 4.1.2) y a su vez permitan establecer el estado de algunos atributos externos del numeral anterior. A continuación, se presenta el proceso de concepción para un modelo de métricas interno que valorará la arquitectura implementada de EVOLUCION 3.5.

De acuerdo con (Vazquez, et al. 2001), la armonización de un modelo de métricas interno exige trabajar simultáneamente elementos en las siguientes cinco (5) dimensiones (ver Figura 5): los compromisos o requisitos que debe cumplir, la forma en que debe medirse (naturaleza), lo que debe medir (atributos), cuando debe medirse (fase o etapa del proceso de desarrollo) y por último con que nivel de detalle (granularidad). La siguiente tabla relaciona de forma breve, cada uno de las dimensiones mostradas en la Figura 5 en correspondencia con sus elementos contenidos. Además se indica por cada elemento, las métricas aplicables en cada uno.

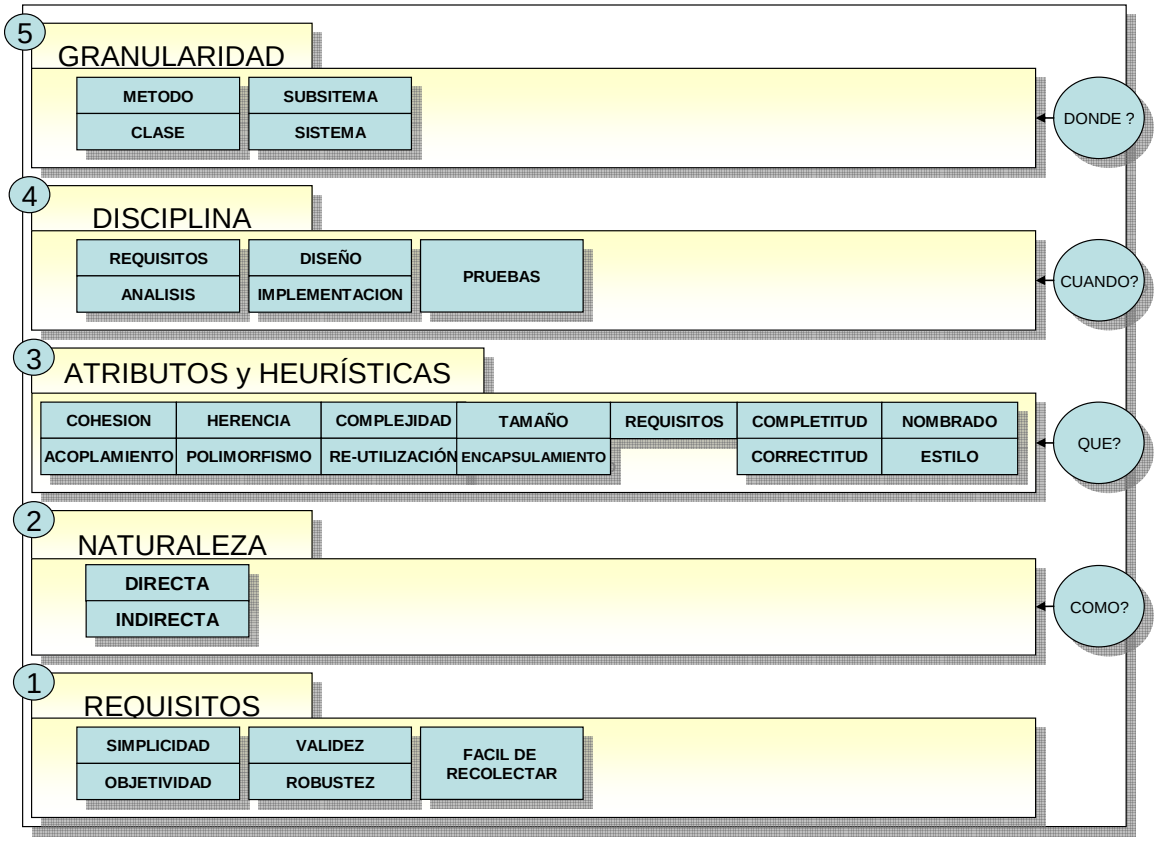

Figura 5: Características de un Modelo de Métricas para EVOLUCION 3.5

DIMENSION	ELEMENTO	DESCRIPCION
REQUISITOS	**SIMPLICIDAD**	La definición y uso de la métrica debe ser simple.
	OBJETIVIDAD	Diferentes personas han de darle valores idénticos. Esto le da consistencia evitando interpretaciones individuales.
	VALIDEZ	El coste y esfuerzo para obtener la medida debe ser razonable.
	ROBUSTEZ	Debe ser insensible a cambios no relevantes, permitiendo realizar comparaciones útiles.
	FACIL RECOLECCION	Debe medir lo que se supone que mide. Esto proporciona fiabilidad.
NATURALEZA	**DIRECTA**	Solo precisan de un único atributo para medirlo.
	INDIRECTA	No miden un atributo directamente. En su lugar, combinan varias métricas directas para medirlo.

DIMENSION	ELEMENTO	DESCRIPCION
ATRIBUTOS	**MODIFICABILIDAD Y DEFECTOS POTENCIALES DE REQUISITOS**	De acuerdo con Kamstiems y Rombach (1997), detectar tempranamente los problemas en la definición de requerimientos es crucial a la hora de mejorar la calidad del proceso de desarrollo. Esto se debe a que el costo de reparar los defectos, se incrementa a medida que el proyecto avanza. Existen dos enfoques para valorar la calidad de los requerimientos: • **Manual:** Este enfoque aborda aspectos relacionados con la estabilidad, ambigüedad y trazabilidad entre requerimientos. • **Automatizado:** Este enfoque utiliza **NPL** (Natural Language Processing) para conocer el vocabulario usado, estilo de redacción, grado de concordancia entre la sintaxis y la semántica del texto, información faltante, oraciones no conectadas entre otros. La modificabilidad es una de las propiedades deseables acerca de la especificación de los requerimientos. IEEE (1993) define una especificación de requerimientos modificable como aquella estructura y estilo de texto que soporta el cambio con facilidad y de forma consistente conservando su esencia. Saeki (2003) propone un conjunto de métricas para valorar la modificabilidad de los diagramas de casos de uso. La idea básica consiste en que si un caso de uso necesita un cambio, probablemente otros casos de uso relacionados también lo necesiten. Las métricas propuestas son: **NOD[3] , NUCT[4]**
		De otra parte, las métricas propuestas por Bernardez et al. (2004) permiten la predicción de defectos potenciales en la especificación de requerimientos que a la larga impactarán la facilidad de mantenimiento a los mismos. Las métricas concebidas para tal propósito son: **NOS[5], NOAS[6], NOSS[7], NOUS[8], NOCS[9], NOE[10], NIE[11], RAAS[12], RASS[13], RUCS[14], CC[15], NAU[16], NMU[17] y NSCU[18].**

[3] Acrónimo de **Number of Dependencies**. (Saeki 2003).
[4] Acrónimo de **Number of Case Use Types**. (Saeki 2003).
[5] Acrónimo de **Number of Steps of the Use Case**. (Bernardez et al., 2004)
[6] Acrónimo de **Number of actor action steps of the use case**. (Bernardez et al., 2004)
[7] Acrónimo de **Number of system action steps ot the use case.** (Bernardez et al., 2004)
[8] Acrónimo de **Number of use case action steps of the use case**. (Bernardez et al., 2004)
[9] Acrónimo de **Number of conditional steps of the use case.** (Bernardez et al , 2004)
[10] Acrónimo de **Number of Exceptions of the use case.** (Bernardez et al. , 2004)
[11] Acrónimo de **Number times the use case is included or extends others.** (Bernardez et al., 2004)
[12] Acrónimo de **Rate of actor action steps of the use case.** (Bernardez et al., 2004)

DIMENSION	ELEMENTO	DESCRIPCION
	COHESION	La cohesión consiste en el grado en que están relacionadas las responsabilidades de una clase o componente. Una alta cohesión indica que un elemento de diseño presenta pocas responsabilidades estrechamente relacionadas, al mismo tiempo que entrega una alta funcionalidad y es fácil de entender por separado. Por otra parte, un elemento con baja cohesión, presenta, muchas responsabilidades no relacionadas, no puede entenderse fácilmente si está aislado y será más difícil de reutilizar y mantener. Las métricas de cohesión están estrechamente relacionadas con las de tamaño. **LCOM**[19] es una métrica para valorar cohesión.

[13] Acrónimo de **Rate of system action steps of the use case.** (Bernardez et al., 2004)
[14] Acrónimo de **Use case action steps of the use case.** (Bernardez et al., 2004)
[15] Acrónimo de **Cyclomatic Complexity of the use cases.** (Bernardez et al., 2004)
[16] Acrónimo de **Number of Actors Associated to a Use Case.** (Kim y Boldyreff., 2002)
[17] Acrónimo de **Number of messages associated with a Use Case.** (Kim y Boldyreff., 2002)
[18] Acrónimo de **Number of System Classes Associated with a Use Case).** (Kim y Boldyreff., 2002)
[19] Acrónimo de **Lack of Cohesion in Methods**. (Chidamber y Kemerer, 1994)

DIMENSION	ELEMENTO	DESCRIPCION
	ACOPLAMIENTO	Consiste en la medida de la dependencia entre los elementos de un diseño. Una alta dependencia, causará necesariamente un impacto sobre la calidad del producto final, específicamente sobre la facilidad de mantenimiento y de prueba. Un buen principio de diseño consiste en minimizar estas dependencias. Las métricas: • **CBO**[20] • **COF**[21] • **Dep_Out**[22] • **Dep_In**[23] • **NumAssEl_ssc**[24] • **NumAssEl_sb**[25] • **NumAssEl_nsb**[26] • **EC_Attr**[27] • **IC_Attr**[28] • **EC_Par**[29] • **IC_Par**[30] • **StimSent**[31] • **StimRecv**[32] • **MsgSent**[33] • **MsgRecv**[34] Están concebidas para valorar el acoplamiento de un sistema y arrojarán como resultado a aquellos elementos de diseño que presentan una alta densidad de fallas.

[20] Acrónimo de **Coupling Between Objects**. (Chidamber y Kemerer, 1994)
[21] Acrónimo de **Coupling Factor**. (Abreu y Melo, 1996).
[22] El número de dependencias (asociaciones) donde la clase es el cliente. (Catálogo de SDMetrics)
[23] El número de dependencias (asociaciones) donde la clase es el proveedor. (Catálogo de SDMetrics)
[24] Para una clase en el paquete ***p***, cuenta únicamente elementos asociados en el mismo paquete ***p.*** (Catálogo de SDMetrics)
[25] Para una clase en el paquete ***p***, cuenta únicamente elementos asociados en los paquetes (contenidos o contenedores) de ***p.*** (Catálogo de SDMetrics)
[26] Para una clase en el paquete ***p***, cuenta únicamente elementos asociados en otros paquetes, **no** contenidos ni contenedores de ***p.*** (Catálogo de SDMetrics)
[27] Número de veces que la clase es usada externamente como un atributo. (Catálogo de SDMetrics)
[28] Número de atributos en la clase que tienen como tipo a otras clases o interfaces. (Briand et. Al 1997)
[29] Número de veces que la clase es utilizada como parámetro. (Briand et. Al 1997).
[30] Número de Parámetros en la clase cuyo tipo es clase o interfaz. (Briand et. Al 1997).
[31] Número de estímulos que los objetos de la clase, envían a objetos de otras. (Briand et. Al 1997).
[32] Número de estímulos que los objetos de la clase reciben de objetos de otras. (Briand et. Al 1997).
[33] Número de mensajes que las instancias de la clase envían a instancias de otras.(Briand et. Al 1997).
[34] Número de mensajes que las instancias de la clase reciben de instancias de otras.(Briand et. Al 1997).

DIMENSION	ELEMENTO	DESCRIPCION
	HERENCIA	La herencia debe valorarse con base en aspectos tales como: Profundidad y anchura del árbol de herencia, número de ancestros y descendientes de un elemento de diseño, polimorfismo, sobre-escritura de métodos entre otros. Por ejemplo: el entendimiento de un elemento de diseño ubicado en un nivel determinado del árbol de herencia, depende del entendimiento de sus ancestros, así mismo una modificación en un elemento de diseño, afectará necesariamente a todos sus descendientes si no se ha previsto la situación. El uso adecuado o inadecuado de la herencia puede afectar a la calidad del producto. Sin embargo, la herencia es poco usada en los elementos de diseño, en otras palabras, pocas clases del sistema estarán involucradas en relaciones de herencia, de aquí que las métricas relacionadas con la herencia, son poco usadas y difíciles de aplicar. Algunas métricas propuestas son : **NumAnc**[35]**, NumDesc**[36], **NOC**[37], **MIF**[38]**, AIF**[39]**, SIX**[40] **y DIT**[41]
	POLIMORFISMO	Esta propiedad indica la posibilidad de que una operación tome muchas formas. Es decir, permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas, según sea el objeto que se referencia en ese momento. La métrica propuesta para valorar polimorfismo es: **POF**[42]

[35] Cuenta el número total de ancestros de la clase. Si no se usa herencia múltiple, entonces arroja el mismo valor que DIT. (Chidamber y Kemerer, 1994).
[36] Cuenta el número total de descendientes de la clase a través de toda una jerarquía. (Lake 1994)
[37] Cuenta el número total directo de descendientes de una clase. (Chidamber y Kemerer, 1994).
[38] Acrónimo de **Method Inheritance Factor**. (Abreu y Melo, 1996).
[39] Acrónimo de **Attribute Inheritance Factor**. (Abreu y Melo, 1996).
[40] Acrónimo de **Specialisation Index per Class**. (Lorentz y Kidd, 1994).
[41] Calcula el camino más largo desde la clase raíz de una jerarquía hasta la clase. (Chidamber y Kemerer, 1994).
[42] Acrónimo de **Polymorphism Factor**. (Abreu y Melo, 1996).

DIMENSION	ELEMENTO	DESCRIPCION
	COMPLEJIDAD	La complejidad mide el grado de conectividad (relaciones/dependencias) entre los elementos de un diseño. Por ejemplo: el número de llamados entre métodos de una misma clase puede considerarse como una medida de la complejidad de la clase. Una alta complejidad, impacta directamente sobre la facilidad de entendimiento del modelo y su posterior esfuerzo para realizarle pruebas. En la práctica, la complejidad esta estrechamente relacionada con el tamaño. Algunas métricas propuestas para valorar la complejidad son: **RFC[43], WMC[44], v(G)[45],** y **MsgSelf [46]**
	REUTILIZACION	El uso correcto de la herencia en los sistemas, permite alcanzar niveles de reutilización deseables. La métrica propuesta para valorar este atributo es: **NOC[47]**
	ENCAPSULAMIENTO	El encapsulamiento (ocultamiento de la información) permite que un objeto pueda acceder a sus datos mediante sus propios métodos al mismo tiempo que los esconde de los demás objetos, esto asegura que los objetos no pueden cambiar el estado interno de otros objetos de maneras inesperadas; solamente los propios métodos internos del objeto pueden acceder a su estado. Las métricas propuestas para valorar el encapsulamiento son: **MHF[48], AHF[49].**

[43] Acrónimo de **Response For a Class**. (Chidamber y Kemerer, 1994).
[44] Acrónimo de **Weighted Methods per Class**. (Chidamber y Kemerer, 1994).
[45] Acrónimo de **Average**. (McCabe 1996).
[46] Cuenta el Número de mensajes que instancias de esta clase envia a sí misma o a instancias de la misma clase. (Lee et. al, 1995)
[47] Acrónimo de **Number of Children**. (Chidamber y Kemerer, 1994)
[48] Acrónimo de **Method Hiding Factor**. (Abreu y Melo, 1996)
[49] Acrónimo de **Attribute Hiding Factor**. (Abreu y Melo, 1996)

DIMENSION	ELEMENTO	DESCRIPCION
	TAMAÑO	Las métricas **(LOC[50], NOM[51], NumAttr[52], NumOps, NumPubOps, Setters y Getters)** orientadas a medir el tamaño del diseño básicamente consisten en el conteo de los elementos contenidos en el modelo. Por ejemplo el número de operaciones en una clase o el número de clases en un paquete. Este tipo de métricas son adecuadas para estimar adecuadamente el coste y esfuerzo para la implementación, revisión, prueba o mantenimiento. Generalmente estas estimaciones son la base para la asignación de personal durante la planificación del proyecto. La facilidad de mantenimiento, reúso y entendimiento son afectados por el número de componentes involucrados en un elemento de diseño. Por último, el tamaño es el principal factor a considerar a la hora de estimar el costo de un proyecto de software.
HEURISTICAS (REGLAS DE DISEÑO)	**CLASE NO USADA**	**La clase no es usada en ninguna parte.** En otras palabras, la clase no tiene clases hijas, dependencias, o asociaciones ni tampoco es usada como parámetro en algún método de otra clase ni como atributo. Probablemente, se necesite modelar los clientes de la clase o considerar borrarla del modelo. Esta regla pertenece a la característica de la COMPLETITUD y su severidad se considera como Alta. (Riel 96) **Nota:** Para modelos que fueron generados a partir del código (reverse engineering), la herramienta SDMetrics puede reportar violaciones falsas a esta regla**.** Esto sucede únicamente para las clases que son referenciadas en la implementación de los métodos a través de variables.
	CLASE NO REPRESENTADA	**La clase del Código no es representada en ninguna parte del modelo.** En otras palabras, la clase que fue hallada mediante la generación del código al modelo, no se encuentra representada en ningún diagrama de la documentación oficial de la herramienta. Esta regla pertenece a la característica de la COMPLETITUD y su severidad se considera como Alta.

[50] Acrónimo de **Lines of Code per Method**. (Chidamber y Kemerer, 1994)
[51] Acrónimo de **Number of Messages Send**. (Lorentz y Kidd, 1994).
[52] Acrónimo de **Number of Attributes for a Class.** (Lorentz y Kidd, 1994)

DIMENSION	ELEMENTO	DESCRIPCION
	PAQUETE NO REPRESENTADO	**El Paquete del Código no es representado en ninguna parte del modelo.** En otras palabras, el paquete que fue hallado mediante la generación del código al modelo, no se encuentra representado en ningún diagrama de la documentación oficial de la herramienta. Esta regla pertenece a la característica de la COMPLETITUD y su severidad se considera como Alta.
	CLASE ENORME	**La Clase es Enorme también conocida como Clase "dios".** Esta situación se presenta cuando una clase tiene entre atributos y operaciones más de 60. Este tipo de clases son un cuello de botella para el mantenimiento del software, también causan problemas de confiabilidad e indican una debilidad en la concepción de una buena arquitectura orientada a objetos. Esta regla pertenece a la categoría de ESTILO y su severidad de considera "Media". (Fowler, 1999)
	OPERACIÓN DUPLICADA	**La Clase tiene operaciones duplicadas.** Consiste en que existen en la clase, dos o más operaciones con signaturas idénticas (nombre de operación y lista de parámetros con sus tipos). Las signaturas de operación deben ser únicas para las clases. Esta regla pertenece a la categoría de CORRECTITUD y su severidad se considera como "Alta". Esta regla está sugerida en los estándares OMG03 y OMG05 que todo modelo UML válido debe cumplir.
	NOMBRE DE CLASE INCORRECTO	**La clase ha sido representada con un nombre que no se corresponde con el código.** En la documentación oficial de la herramienta el nombre de la clase es diferente al nombre real que fue encontrado en el código fuente de la misma. Esta regla pertenece a la categoría de CORRECTITUD y su severidad de considera como "Media".
	REFERENCIAS CIRCULARES	**La clase tiene referencias circulares.** Las dependencias circulares deberían evitarse. Las clases que participan en el ciclo, no pueden probarse ni re-usarse de forma independiente. Entre más clases participen en el ciclo, peor es el problema, especialmente si las clases residen en diferentes paquetes. Debe revisarse el diseño para eliminar el ciclo de dependencia. Esta regla pertenece a la categoría de "ESTILO" y su severidad se considera "Media". (SDMetrics – Measurement Catalog)

DIMENSION	ELEMENTO	DESCRIPCION
	SOBRE-ESCRITURA DE ATRIBUTO HEREDADO	**La clase define un atributo del mismo nombre que un atributo heredado.** Durante la generación de código, esta situación puede ocultar inadvertidamente el atributo de la clase padre. Debería cambiarse el nombre del atributo en la clase hija. Esta regla pertenece a la categoría de NOMBRADO y su severidad se considera como "Media". (Ramírez et. al, 2004)
DISCIPLINA	**REQUISITOS**	**Métricas Aplicables:** NOD, NUCT, NOS, NOAS, NOSS, NOUS, NOCS, NOE, NIE, NOAS/NOS, NOSS/NOS, NOUS/NOS y CC.
	DISEÑO	**Métricas Aplicables:** CBO, COF, Dep_Out, Dep_In, NumAssEL_ssc, NumAssEL_sb, NumAssEL_nsb, EC_Attr, IC_Attr, IC_Par, StimSent, StimRecv, MsgSent, MsgRecv, NumAnc, NumDesc, MsgSelf, LCOM, RFC, WMC, MIF, SIX, DIT, POF y NOC.
	IMPLEMENTACION	**Métricas Aplicables:** LCOM, v(G), MHF, AHF, MIF, SIX, POF, y LOC.
	PRUEBAS	**Métricas Aplicables:** ICC[53], SCC[54] y UCC[55].
GRANULARIDAD	**METODO**	**Las métricas aplicables en este contexto son:** CBO, COF, RFC, WMC, MsgSelf, v(G), MIF, AIF, LOC y NOM.
	CLASE	**Las métricas aplicables en este contexto son:** CBO, COF, RFC, WMC, v(G), MHF, AHF, MIF, AIF, SIX, DIT, POF, NOC, LOC, Dep_Out, Dep_In, NumAssEL_ssc, NumAssEL_sb, NumAssEL_nsb, EC_Attr, IC_Attr, IC_Par, StimSent, StimRecv, MsgSent, MsgRecv, NumAnc, NumDesc.
	PAQUETE	**Las métricas aplicables en este contexto son:** CBO, COF, RFC, WMC, v(G), MIF, AIF y LOC.
	SISTEMA	**Las métricas aplicables en este contexto son:** CBO, COF, RFC, WMC, v(G), MIF, AIF y LOC
	MODELO DE CASOS DE USO	**Las métricas aplicables en este contexto son:** NOD, NUCT, NOS, NOAS, NOSS, NOUS, NOCS, NOE, NIE, NOAS/NOS, NOSS/NOS, NOUS/NOS y CC.

Tabla 3: Atributos de Calidad Internos

[53] Acrónimo de **Inhheritance Context Coverage.** (IPL 1999)
[54] Acrónimo de **State-based Context Coverage.** (IPL 1999)
[55] Acrónimo de **Multi-trheaded Context Coverage.** (IPL 1999)

En el Anexo 1**,** se describe brevemente las métricas propuestas para evaluar la arquitectura de software de la herramienta **EVOLUCION 3.5.** Las métricas se organizan por atributos internos de la arquitectura (tamaño, acoplamiento, cohesión, encapsulamiento, herencia, polimorfismo y requisitos). Adicionalmente cada métrica está acompañada de su definición, valoración y autor(es)**.**

En el siguiente numeral, se enuncian las herramientas utilizadas para apoyar la evaluación arquitectónica de **EVOLUCION 3.5**. Para cada herramienta se provee una breve descripción así como una ilustración de su interfaz gráfica de usuario.

4.1.4. Exploración y Selección de Herramientas disponibles

Durante el desarrollo de la evaluación arquitectónica, se exploró en diversas fuentes (como Internet) en busca de herramientas que pudieran llevar a cabo parcial o totalmente la evaluación de todos los artefactos (requisitos, diagramas, modelos y código) disponibles para la herramienta EVOLUCION 3.5. A continuación se provee una breve descripción de las herramientas encontradas y seleccionadas para apoyar la evaluación arquitectónica.

- **UMLStudio [20].** Esta herramienta ayuda a los desarrolladores de software a ser más sistemáticos y productivos en sus proyectos. Por medio del modelado, los desarrolladores pueden visualizar, analizar y comunicar sus ideas de forma más efectiva en el marco de sistemas de software grandes y complejos. Además de los modelos gráficos, esta herramienta permite la generación automática de código fuente genérico que luego puede ser completado y adecuado a las necesidades de los equipos de desarrollo. Esta herramienta CASE provee las siguientes capacidades: Uno o más editores para la creación de diagramas en notación UML, características de validación sintáctica y semántica de los diagramas, un generador de código automático y capacidad de ingeniería inversa y un repositorio en el cuál los diagramas y sus atributos puedan almacenarse y recuperarse. Todos los

modelos entregados por SIMON, están elaborados en esta herramienta. La Figura 6 muestra el aspecto de esta herramienta. Como limitaciones de la versión usada pueden encontrarse las siguientes:

- Las capacidades de importación/exportación están restringidas.
- De igual forma la herramienta sólo permite abrir ficheros propietarios a diferencia de otras que aceptan formatos como XMI y XML.
- Sus capacidades de ingeniería inversa, no incluyen a Delphi.

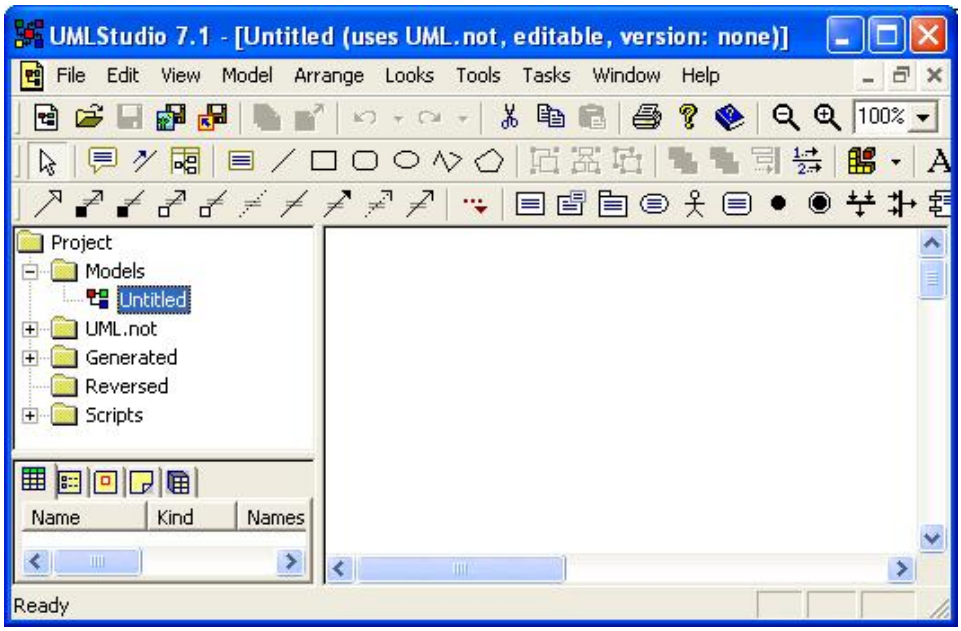

Figura 6: Interfaz de UMLStudio

SDMetrics[10]. Esta herramienta provee algunas capacidades automáticas para analizar y valorar un modelo ya generado a partir de código fuente con relación a un conjunto de métricas de software y heurísticas de diseño. Necesita como entrada un archivo en formato XMI y como salida provee tres vistas: Métrica, Elemento y Tabla. Las métricas que es capaz de evaluar están agrupadas en: Tamaño, Acoplamiento, Complejidad y Herencia. De otra parte las heurísticas de diseño

valoradas se agrupan en: Completitud, Correctitud, Nombrado y Estilo entre otras. Debido a que la versión utilizada es una ***"Demo"***, las capacidades de exportación de datos a otros formatos está restringida, por lo tanto el copiado y organización de estos últimos tuvo que hacerse manualmente. La Figura 7 muestra el aspecto de la herramienta.

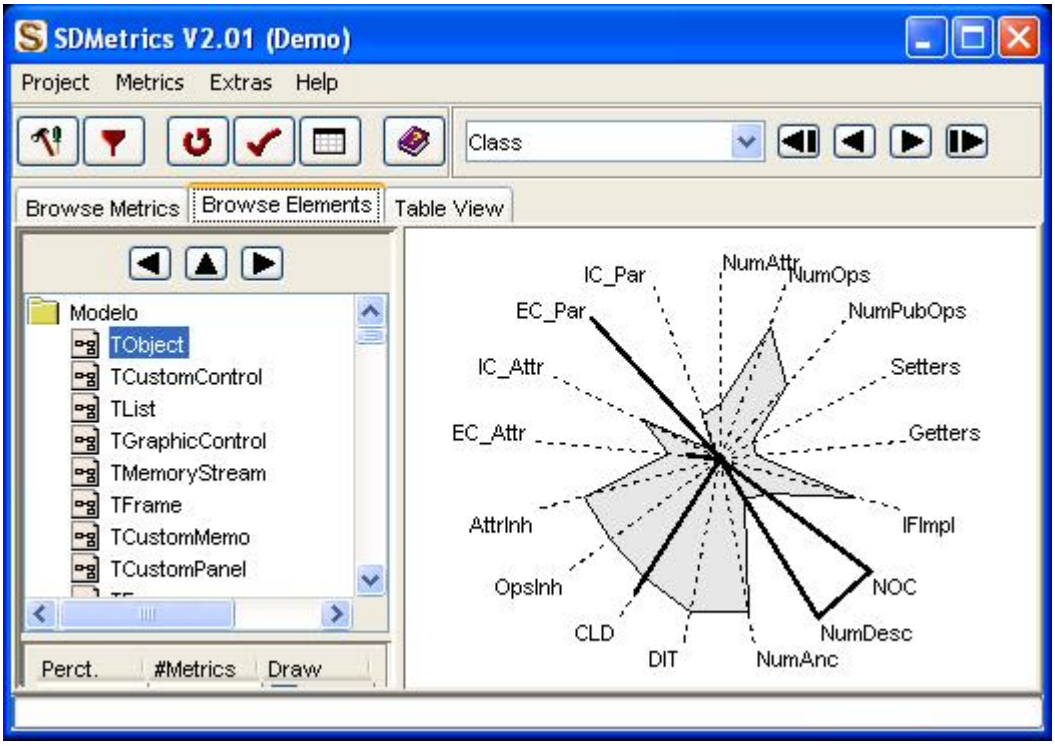

Figura 7: Interfaz de SDMetrics

EssModel[2]. Esta herramienta permite la generación de un archivo en formato XMI a partir de código fuente escrito en C, Java o Delphi. Por otra parte, permite la navegación y edición a través del modelo generado a partir del código fuente con el propósito de crear la documentación del mismo en formato de HTML. El uso de este software fue indispensable debido a que es capaz de generar el archivo XMI que necesita **SDMetrics** para evaluar al código fuente, además fue bastante útil a la

hora de calcular algunas métricas importantes de forma manual que **SDMetrics** no provee. La Figura 8 muestra el aspecto de la herramienta.

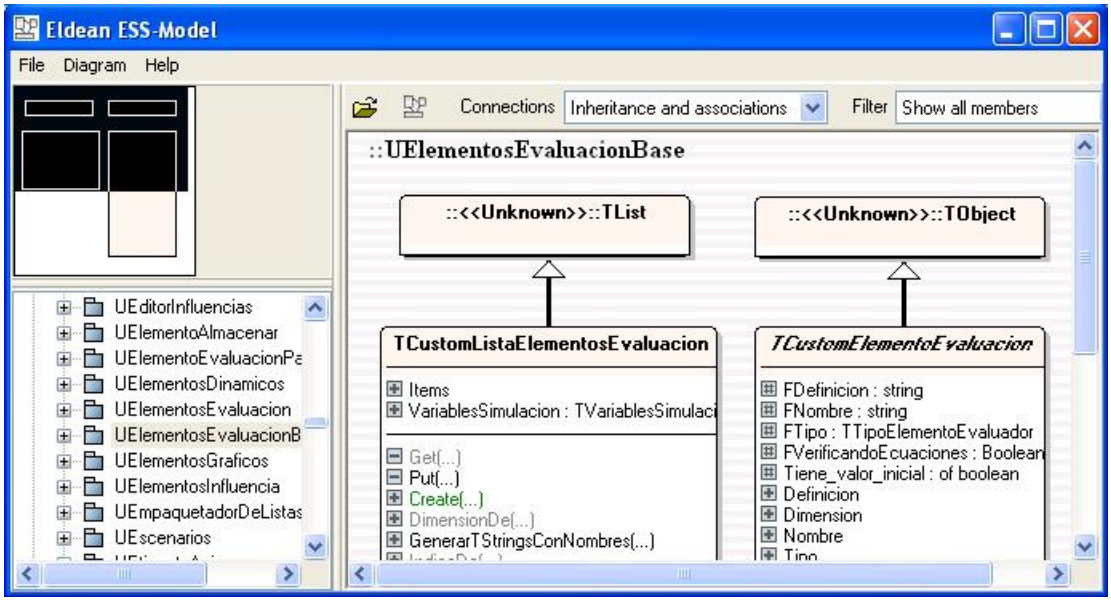

Figura 8: Interfaz de EssModel

Microsoft Excel[7]. Esta herramienta sirvió para la tabulación, organización e interpretación de los datos derivados de la evaluación arquitectónica realizada sobre EVOLUCION 3.5. La Figura 9 muestra el aspecto de la herramienta.

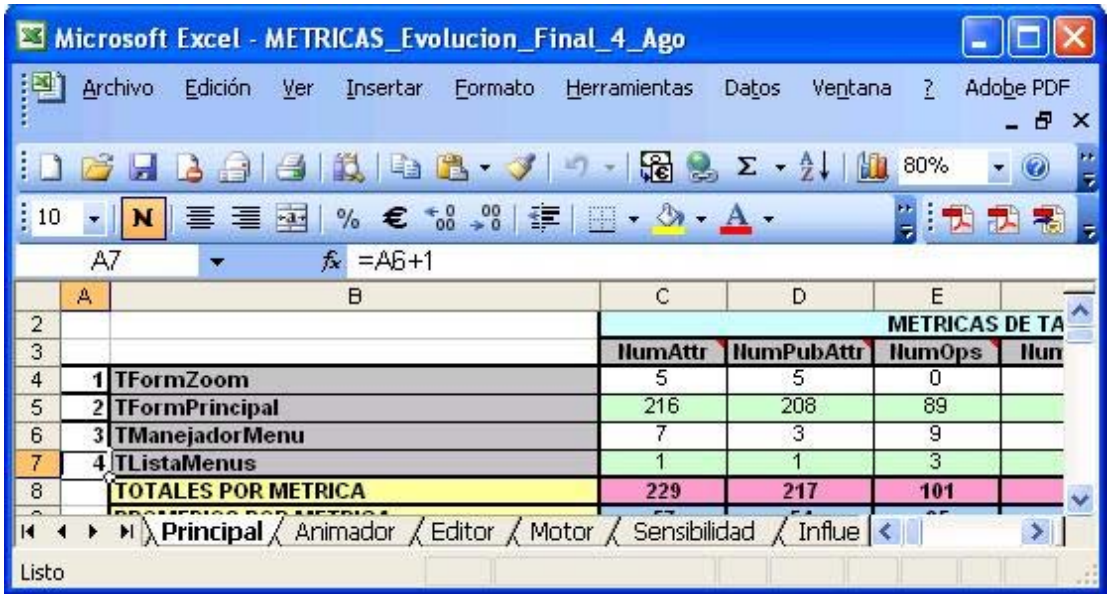

Figura 9: Interfaz de Microsoft Excel

4.1.5. Elaboración del Modelo de Datos para la Evaluación

Una vez reconocido el universo de artefactos disponible (requisitos, diagramas y código) y las métricas que se aplicarían sobre estos, se determinó un diseño de datos simple sobre la herramienta Microsoft Excel que permitiera la tabulación de los valores de cada métrica por cada elemento de evaluación (caso de uso, clase, paquete e interfaz). A continuación en la Figura 10 se ilustra el modelo de datos que subyace en el libro de Microsoft Excel "Métricas _ Evolución.xls".

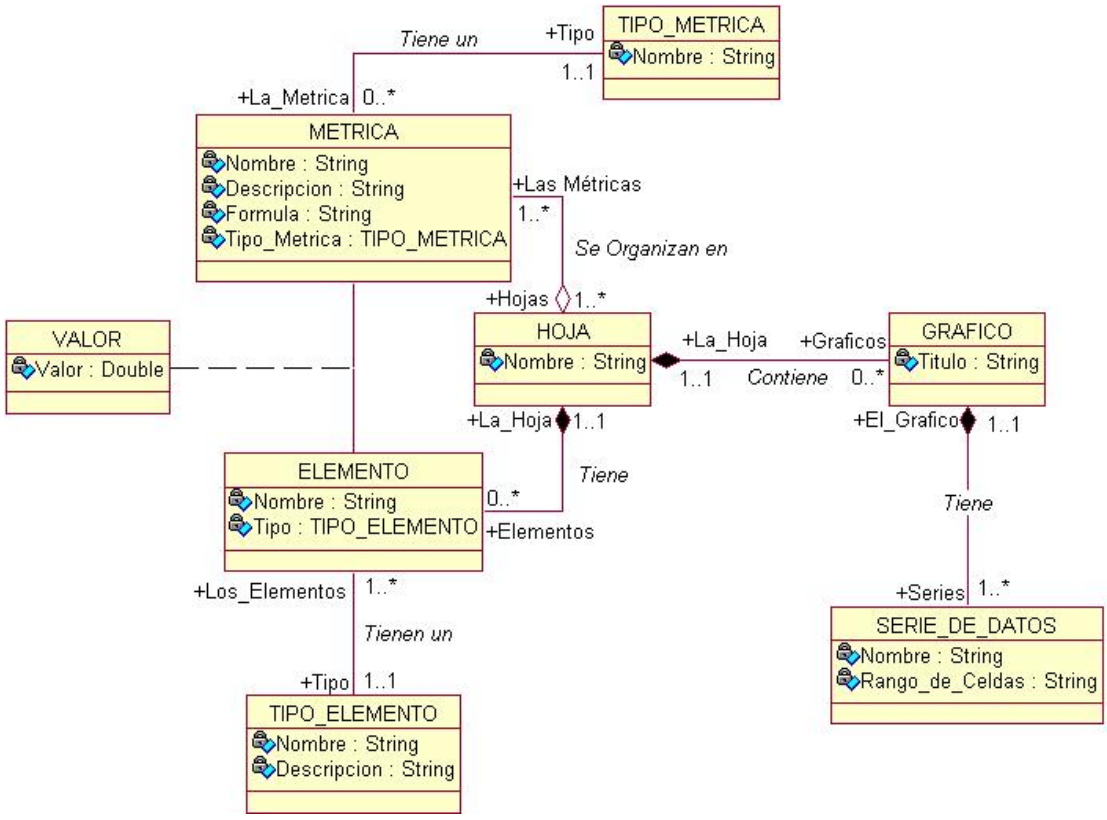

Figura 10: Modelo de Datos para la Hoja de Excel "Metricas_Evolucion.xls"

En la Tabla 4 se provee una breve descripción de cada concepto mostrado en la Figura 10.

CONCEPTO	DESCRIPCION
TIPO__METRICA	Las métricas calculadas pertenecen a varias categorías: Tamaño, Acoplamiento, Herencia, Complejidad etc. En el archivo "Metricas_Evolución.xls" se agruparán visualmente las métricas según su tipo (Tamaño, Herencia, Acoplamiento etc.).
METRICA	Corresponde a la métrica que se valora, la cuál se describe por su acrónimo, descripción (motivación y autor) y su fórmula. Cada métrica es calculada para todos y cada uno de los elementos según corresponda.
ELEMENTO	Corresponde al objeto evaluado, el cuál puede ser una clase, un paquete, un caso de uso, entre otros. Los elementos se agrupan por módulos u hojas de cálculo en el libro "Metricas_Evolución.xls"
TIPO_ELEMENTO	Indica su naturaleza: Clase, Diagrama, Caso de Uso, Paquete entre otros.

VALOR	Almacena el resultado final de la fórmula de cada métrica por cada elemento evaluado.
HOJA	Corresponde a las hojas del libro "Metricas_Evolución.xls". Cada hoja, almacena un conjunto de métricas y elementos relacionados, aunque es posible que existan solo hojas de resultados. Se crearon alrededor de 21 hojas de cálculo en el archivo Metricas_Evolución.xls, e igual cantidad de hojas de gráficos en el archivo Graficos_Evolucion.xls.
GRAFICO	Se incrustan en las hojas con el objeto de resumir y consolidar visualmente los datos tabulados. Se proveen tres tipos de perspectivas en los gráficos: Comparación entre métricas del mismo tipo, comparación de métricas de todos los tipos para elementos del mismo módulo y comparación entre atributos para el módulo. Los gráficos usados son del tipo *barras horizontales* con el propósito de facilitar la comparación. Además se incluyeron otros gráficos de más alto nivel con el objeto de soportar las conclusiones arrojadas. Todos los gráficos se encuentran en el archivo Graficos_Evolucion.xls
SERIE_DATOS	Cada gráfico se *alimenta* de un conjunto de datos que puede agruparse por series, las cuáles son representadas por distintos colores con el objeto de diferenciarlas unas de otras.

Tabla 4: Elementos del Modelo de Datos de Evaluación Arquitectónica

Finalmente es importante destacar que aunque el código fuente no refleja en empaquetamiento sugerido por el modelo elaborado en la herramienta **UMLStudio,** ni el documento de tesis "EVOLUCIÓN 3.5. HERRAMIENTA SOFTWARE PARA EL MODELAMIENTO Y SIMULACIÓN CON DINÁMICA DE SISTEMAS", los archivos Metricas_Evolución.xls y Graficos_Evolucion.xls sí reflejan esta estructura con el objeto de respetar y mantener el espíritu original de los autores en la concepción de **EVOLUCION 3.5,** aspecto que hará más comprensible a los desarrolladores de SIMON interpretar los datos de la evaluación arquitectónica aquí presentada.

4.1.6. Aplicación de la Evaluación

La evaluación arquitectónica se constituyó a partir de tres valoraciones: automática, semi-automática y manual. La primera se llevo a cabo por medio de la herramienta **SDMetrics**, la segunda mediante la herramienta **Ms-Excel** y la tercera a través de **EssModel**. El procedimiento para realizar la valoración automática fue el siguiente:

1. Abrir el directorio "***Source_Code***[56]" mediante el comando "**Open Folder**" de la herramienta **EssModel**.

2. Generar el archivo en formato **XMI** a partir del código fuente de **EVOLUCION 3.5** mediante el comando "**Export Model to a XMI-File**" de la herramienta **EssModel**. El archivo generado tuvo un tamaño de 12 MB, arrojando un total de 448 clases y 239 paquetes.

3. Abrir el archivo **XMI** generado en el paso anterior y ejecutar el comando "***Calculate Metrics***" en la herramienta **SDMetrics**.

4. Copiar manualmente[57] cada fila de valores de métricas por cada clase del código fuente de **EVOLUCION 3.5**, en el archivo "Metricas_Evolución.xls" según corresponda.

Por otra parte, el procedimiento para valorar manualmente el estado de algunas métricas no contempladas por **SDMetrics** fue el siguiente:

1. Abrir el directorio "***Source_Code***" en el comando "**Open Folder**" de la herramienta **EssModel**.

2. Recorrer manualmente el árbol de paquetes o clases que provee **EssModel** con el propósito de buscar la información necesaria para calcular la métrica. Generalmente esta información se deriva a partir del conteo manual de algunos aspectos clave involucrados en la fórmula de la métrica.

3. Consignar el resultado en el archivo "Metricas_Evolución.xls" según corresponda.

---

[56] Esta carpeta contiene todo el código fuente de EVOLUCION 3.5.
[57] El copiado manual de los datos se requirió debido a que la versión de SDMetrics fue una Demo.

4.1.7. Consolidación e Interpretación de Resultados

Una vez finalizada la evaluación de las métricas del modelo propuesto en el numeral 4.1.3.2 usando toda la información disponible (numeral 4.1.2) y las herramientas descritas en el numeral 4.1.4, se alcanzó el diligenciamiento completo del archivo Metricas_Evolución.xls cuyo modelo de datos se especificó en el numeral 4.1.5. A continuación se describe la interpretación de datos relacionados con la evaluación arquitectónica de **EVOLUCION 3.5.**

4.1.7.1. Ficha Técnica.

En total se recopilaron 31.503 datos agrupados en 2.184 registros y distribuidos en 20 hojas de cálculo de excel. **SDMetrics** calculó de forma automática 15.478 datos (49,13% del total), mientras que 5.987 datos (19% del total) fueron calculados por **Ms-Excel** de forma semi-automática. Por último, 10.038 datos (31,86% del total) se contabilizaron manualmente con la ayuda de la herramienta **EssModel.** La Figura 11[58] ilustra esta situación, a la luz de los aspectos y formas de registro de métricas para **EVOLUCION 3.5**. El esfuerzo total en horas/hombre para la realización de la evaluación e interpretación de resultados, arrojó un valor de 287 horas/hombre. Esta información es útil a la hora de estimar futuros esfuerzos de evaluación arquitectónica y/o de justificar un desarrollo orientado hacia esta área de la ingeniería del software.

[58] Cuya fuente de datos es el archivo Metricas_Evolucion.xls en su hoja “Ficha”.

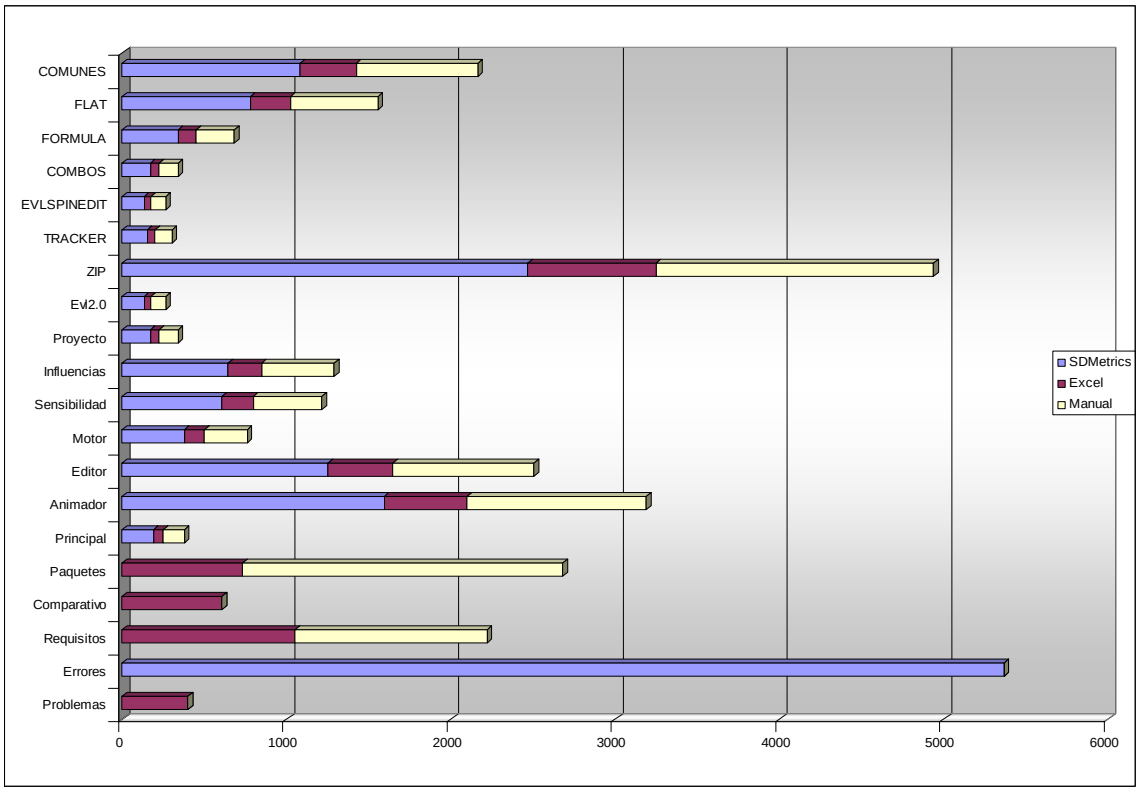

Figura 11: Cantidad de Datos Recopilados por Método y por Aspecto

Por otra parte, en la Figura 12, se muestra la cantidad de datos registrado por cada uno de los aspectos lógicos de la evaluación arquitectónica de **EVOLUCION 3.5** , allí se puede notar que la mayor densidad de datos se encuentra concentrada en los aspectos “Errores”, “ZIP”, “Animador”, “Editor” y “Paquetes”.

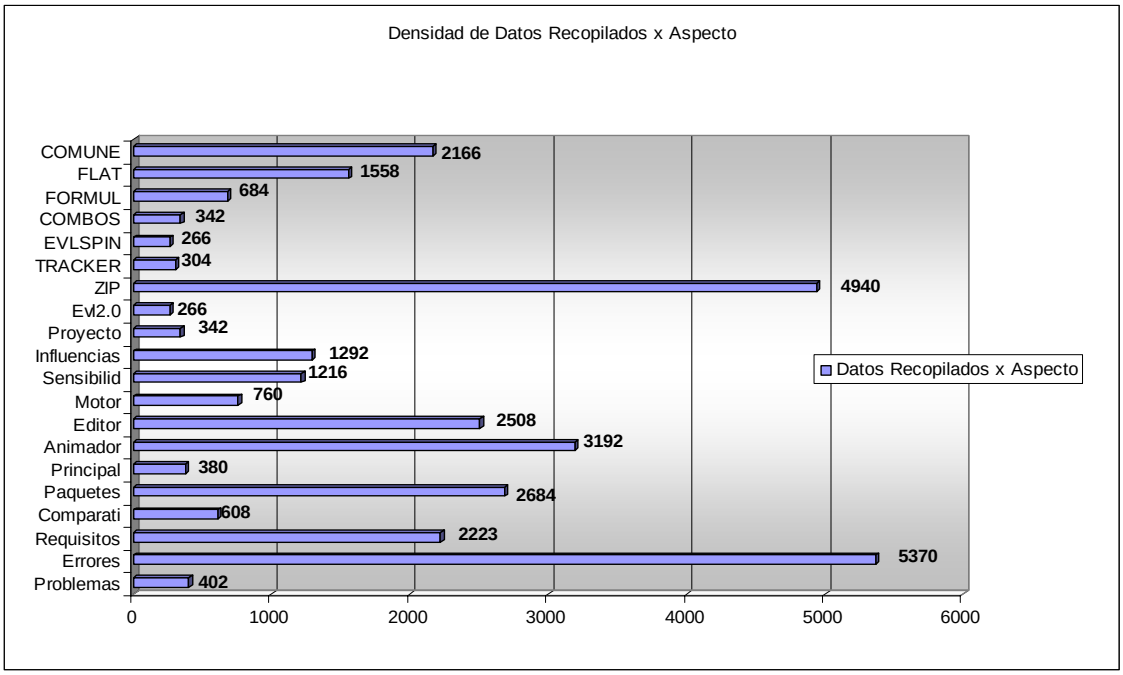

Figura 12: Densidad de Datos por Aspecto

Finalmente, en la Tabla 5, se muestra la cantidad de datos recopilados (automática, semi-automática o manualmente) por cada uno de los 20 aspectos contemplados en la evaluación arquitectónica de **EVOLUCION 3.5.**

Aspecto	SDMetrics	Ms-Excel	Manual	Total de Datos por Aspecto	Proporción densidad de Datos por Aspecto
Problemas	0	402	0	**402**	**1%**
Errores	5370	0	0	**5370**	**17%**
Requisitos	0	1053	1170	**2223**	**7%**
Comparativo	0	608	0	**608**	**2%**
Paquetes	0	732	1952	**2684**	**9%**
Principal	190	60	130	**380**	**1%**
Animador	1596	504	1092	**3192**	**10%**
Editor	1254	396	858	**2508**	**8%**
Motor	380	120	260	**760**	**2%**
Sensibilidad	608	192	416	**1216**	**4%**
Influencias	646	204	442	**1292**	**4%**

Aspecto	SDMetrics	Ms-Excel	Manual	Total de Datos por Aspecto	Proporción densidad de Datos por Aspecto
Proyecto	171	54	117	**342**	**1%**
Evl2.0	133	42	91	**266**	**1%**
ZIP	2470	780	1690	**4940**	**16%**
TRACKER	152	48	104	**304**	**1%**
EVLSPINEDIT	133	42	91	**266**	**1%**
COMBOS	171	54	117	**342**	**1%**
FORMULA	342	108	234	**684**	**2%**
FLAT	779	246	533	**1558**	**5%**
COMUNES	1083	342	741	**2166**	**7%**

Tabla 5: Datos recopilados y evaluados por Aspecto

4.1.7.2. Organización del Informe de Evaluación Arquitectónica

El informe de evaluación arquitectónica para EVOLUCION 3.5 se compone de dos archivos de Ms-Excel mutuamente relacionados Metricas_Evolucion.xls y Graficos_Evolucion.xls. A continuación se describe el contenido del archivo Metricas_Evolucion.xls mediante una tabla que especifica el propósito, número de datos, registros, métricas y gráficos que corresponden a cada módulo de la herramienta EVOLUCION 3.5. En cada caso se proveen enlaces a los directorios o documentos según corresponda.

Hoja	Descripción
Problemas	El propósito fundamental de esta hoja consiste en enumerar y categorizar los problemas relacionados con las heurísticas (COMPLETITUD, CORRECTITUD, NOMBRADO y ESTILO) de diseño para modelos orientados a objetos y escritos en UML. Además muestra el estado final de los factores de Acoplamiento, Herencia y Encapsulamiento para el sistema. En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 4 gráficos.
	Número de Datos: 402 corresponde al 1% del total
	Número de Registros: 201
	Número de Datos Calculados Automáticamente: 0
	Número de Datos Calculados Semi-Automáticamente: 402
	Número de Datos Calculados Manualmente: 0
Comparativo	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño,

Hoja	Descripción
	herencia, encapsulamiento y complejidad para todos los módulos de la herramienta EVOLUCION 3.5, permitiendo realizar una comparación del estado de cualquier módulo respecto a los demás. En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 10 gráficos.
	Número de Datos: 608 corresponde al 2% del total
	Número de Registros: 16
	Número de Datos Calculados Automáticamente: 0
	Número de Datos Calculados Semi-Automáticamente: 608
	Número de Datos Calculados Manualmente: 0
Errores	El propósito fundamental de esta hoja consiste en enumerar para todos los módulos de la herramienta EVOLUCION 3.5 las violaciones a las reglas de buen diseño o heurísticas para COMPLETITUD, CORRECTITUD, NOMBRADO y ESTILO.
	Número de Datos: 5370 corresponde al 17% del total
	Número de Registros: 1074
	Número de Datos Calculados Automáticamente: 5370
	Número de Datos Calculados Semi-Automáticamente: 0
	Número de Datos Calculados Manualmente: 0
Requisitos	El propósito fundamental de esta hoja consiste en mostrar el estado final de los las métricas relacionadas con los casos de uso que se proporcionaron en la documentación oficial de la herramienta EVOLUCION 3.5, permitiendo valorar el estado del modelo de requisitos. Se calcularon 19 métricas.
	Número de Datos: 2223 corresponde al 7% del total
	Número de Registros: 117
	Número de Datos Calculados Automáticamente: 0
	Número de Datos Calculados Semi-Automáticamente: 1053
	Número de Datos Calculados Manualmente: 1170
Paquetes	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todos los paquetes de la herramienta EVOLUCION 3.5, permitiendo realizar una comparación del estado de cualquier paquete respecto a los demás. Se calcularon 11 métricas.
	Número de Datos: 2684 corresponde al 9% del total
	Número de Registros: 244
	Número de Datos Calculados Automáticamente: 0
	Número de Datos Calculados Semi-Automáticamente: 732
	Número de Datos Calculados Manualmente: 1952
Principal	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en el archivo "UPrincipal.pas". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 380 corresponde al 1% del total
	Número de Registros: 10

Hoja	Descripción
	Número de Datos Calculados Automáticamente: 190
	Número de Datos Calculados Semi-Automáticamente: 60
	Número de Datos Calculados Manualmente: 130
Animador	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "Animador". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 3192 corresponde al 10% del total
	Número de Registros: 84
	Número de Datos Calculados Automáticamente: 1596
	Número de Datos Calculados Semi-Automáticamente: 504
	Número de Datos Calculados Manualmente: 1092
Editor	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "Editor". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 2508 corresponde al 8% del total
	Número de Registros: 66
	Número de Datos Calculados Automáticamente: 1254
	Número de Datos Calculados Semi-Automáticamente: 396
	Número de Datos Calculados Manualmente: 858
Motor	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "Motor". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 760 corresponde al 2% del total
	Número de Registros: 20
	Número de Datos Calculados Automáticamente: 380
	Número de Datos Calculados Semi-Automáticamente: 120
	Número de Datos Calculados Manualmente: 260
Sensibilidad	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "Sensibilidad". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 1216 corresponde al 4% del total
	Número de Registros: 32
	Número de Datos Calculados Automáticamente: 608
	Número de Datos Calculados Semi-Automáticamente: 192
	Número de Datos Calculados Manualmente: 416

Hoja	Descripción
Influencias	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "Influencias". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 1292 corresponde al 4% del total
	Número de Registros: 34
	Número de Datos Calculados Automáticamente: 646
	Número de Datos Calculados Semi-Automáticamente: 204
	Número de Datos Calculados Manualmente: 442
Proyecto	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "Proyecto". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 342 corresponde al 1% del total
	Número de Registros: 9
	Número de Datos Calculados Automáticamente: 171
	Número de Datos Calculados Semi-Automáticamente: 54
	Número de Datos Calculados Manualmente: 117
Evl2.0	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "Evl2.0". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 266 corresponde al 1% del total
	Número de Registros: 7
	Número de Datos Calculados Automáticamente: 133
	Número de Datos Calculados Semi-Automáticamente: 42
	Número de Datos Calculados Manualmente: 91
ZIP	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "ZIP". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 4940 corresponde al 16% del total
	Número de Registros: 130
	Número de Datos Calculados Automáticamente: 2470
	Número de Datos Calculados Semi-Automáticamente: 780
	Número de Datos Calculados Manualmente: 1690
TRACKER	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la

Hoja	Descripción
	herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "TRACKER". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 304 corresponde al 1% del total
	Número de Registros: 8
	Número de Datos Calculados Automáticamente: 152
	Número de Datos Calculados Semi-Automáticamente: 48
	Número de Datos Calculados Manualmente: 104
EVLSPINEDIT	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "SpinEdit". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 266 corresponde al 1% del total
	Número de Registros: 7
	Número de Datos Calculados Automáticamente: 133
	Número de Datos Calculados Semi-Automáticamente: 42
	Número de Datos Calculados Manualmente: 91
COMBOS	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "Combos". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 342 corresponde al 1% del total
	Número de Registros: 9
	Número de Datos Calculados Automáticamente: 171
	Número de Datos Calculados Semi-Automáticamente: 54
	Número de Datos Calculados Manualmente: 117
FORMULA	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "FC". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 684 corresponde al 2% del total
	Número de Registros: 18
	Número de Datos Calculados Automáticamente: 342
	Número de Datos Calculados Semi-Automáticamente: 108
	Número de Datos Calculados Manualmente: 234
FLAT	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "Flatstyl". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.

Hoja	Descripción
	Número de Datos: 1558 corresponde al 5% del total
	Número de Registros: 41
	Número de Datos Calculados Automáticamente: 779
	Número de Datos Calculados Semi-Automáticamente: 246
	Número de Datos Calculados Manualmente: 533
COMUNES	El propósito fundamental de esta hoja consiste en mostrar el estado final de los grupos de métricas relacionadas con el acoplamiento, tamaño, herencia, encapsulamiento y complejidad para todas las clases de la herramienta EVOLUCION 3.5, que se agruparon en los archivos ".pas" del directorio "Comunes". En el archivo Graficos_Evolucion.xls se ilustran estos datos mediante 6 gráficos. Se calcularon 38 métricas.
	Número de Datos: 2166 corresponde al 7% del total
	Número de Registros: 57
	Número de Datos Calculados Automáticamente: 1083
	Número de Datos Calculados Semi-Automáticamente: 342
	Número de Datos Calculados Manualmente: 741

Tabla 6: Descripción del Informe de Evaluación Arquitectónica

4.1.8. Elaboración de un compendio de Fallos Arquitectónicos

Luego de exponer la estructura del informe de evaluación arquitectónica, se describe a continuación, un compendio de los errores o fallos (violaciones a las buenas prácticas de diseño propuestas en las heurísticas o reglas descritas en la **Tabla 3: Atributos de Calidad Internos)** encontrados al final del proceso de evaluación**.** Cada violación a la regla estará acompañada de una recomendación general para corregirla en el numeral 4.1.9 Además con el propósito de simplificar la lectura de este documento, se ha compilado el mismo compendio de fallos más extendido en la hoja "Compendio" del Archivo "Metricas_Evolucion.xls". Los aspectos más destacados relacionados con violaciones a las reglas o heurísticas se enuncian a continuación:

- **COMPLETITUD.** En total 4081 problemas encontrados.

  a) Del 100% de los paquetes encontrados en el código fuente, ninguno fue representado en el modelo de la documentación oficial de EVOLUCION 3.5 afectando la COMPLETITUD y CORRECTITUD del sistema en este

aspecto. En total 239 de los 239 paquetes presentan esta violación a la regla.

b) Además el 63% de las clases encontradas en el código fuente no están representadas en el modelo de **UMLStudio** que entregó SIMON. En total 282 de las 448 clases presentan esta violación a la regla.

c) Lo anterior se agrava al determinar que sólo el 15% de la documentación UML (casos de uso y diagramas de secuencia) del sistema está completa. De lo que apenas existe se encontraron 158 problemas de especificación y completitud para casos de uso. Además el modelo de interacción está incompleto en al menos un 88% arrojando 3176 problemas de completitud.

d) El 50% de las clases en el código, aparentemente no se usan. Esto se debe a que en estos casos, las clases son usadas como tipos de parámetros o variables en procedimientos de otras. En total 226 de las 448 clases presentan esta violación aparente a la regla.

e) La distribución de violaciones a la COMPLETITUD discriminada por módulos se ilustra a continuación en la Figura 13. Puede observarse una marcada carencia de COMPLETITUD en los módulos: animador, editor, influencias, motor y comunes. Así mismo los componentes que más aportan a la degradación de la COMPLETITUD, son los componentes ZIP y FLAT.

f) Los datos de NO-Completitud discriminados por cada módulo se presentan a continuación:

Módulo	**Num. Violaciones a la COMPLETITUD**
Principal	30
Animador	762
Editor	362

Módulo	Num. Violaciones a la COMPLETITUD
Motor	146
Sensibilidad	277
Influencias	331
Proyecto	12
EVL2.0	10
ZIP	1239
Tracker	11
EVLSpinEdit	10
Combos	29
Formula	104
Flat	359
Comunes	399

Tabla 7: Distribución de Problemas de Completitud x Módulo

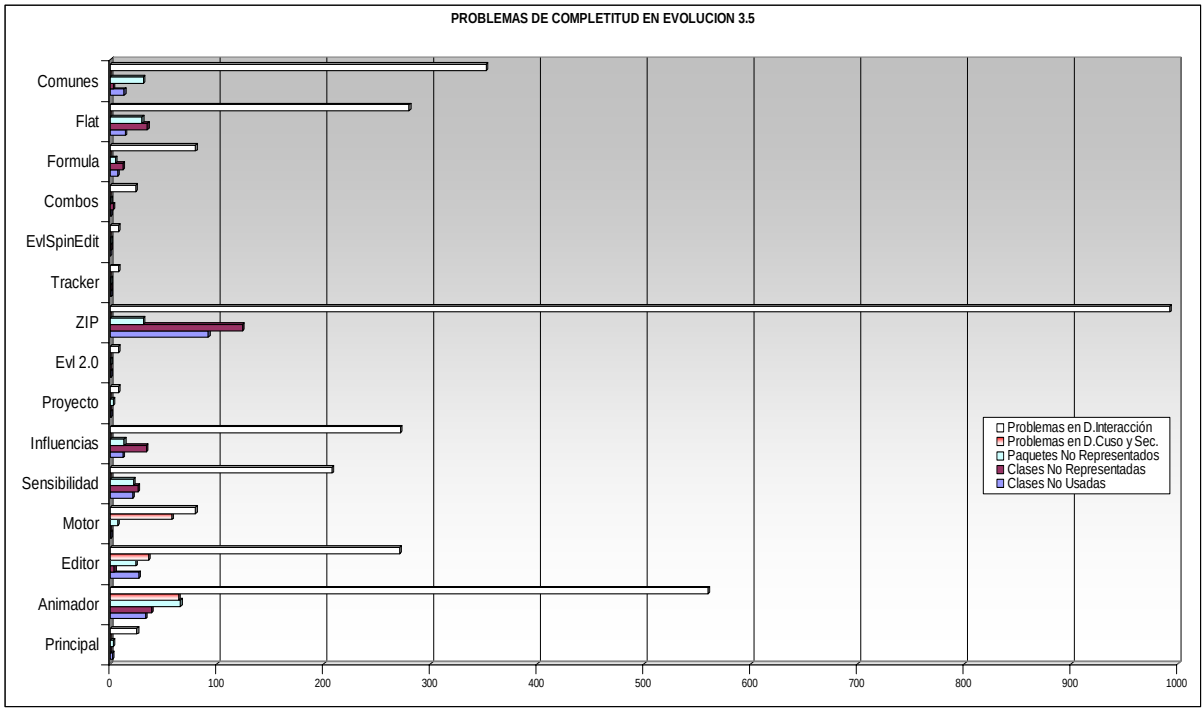

Figura 13: Distribución de Fallos de COMPLETITUD x Módulo

- **CORRECTITUD.** En total 25 problemas encontrados.

a) El 5% de las clases encontradas en el código fuente, presentan un nombre incorrecto en relación con los diagramas de clase provistos en la documentación oficial de EVOLUCION 3.5. En total 23 de las 448 clases presentan esta violación a la regla.

b) Mientras que el 0.45% del total de clases tiene operaciones duplicadas. En total 2 de las 448 clases presentan esta violación a la regla.

c) La distribución de violaciones a la CORRECTITUD discriminada por módulos se ilustra a continuación en la Figura 14. Puede observarse una marcada carencia de CORRECTITUD en los módulos: ***principal, animador, editor, proyecto y comunes***, en estos módulos la ambigüedad nominal entre el código y el modelo es muy alta.

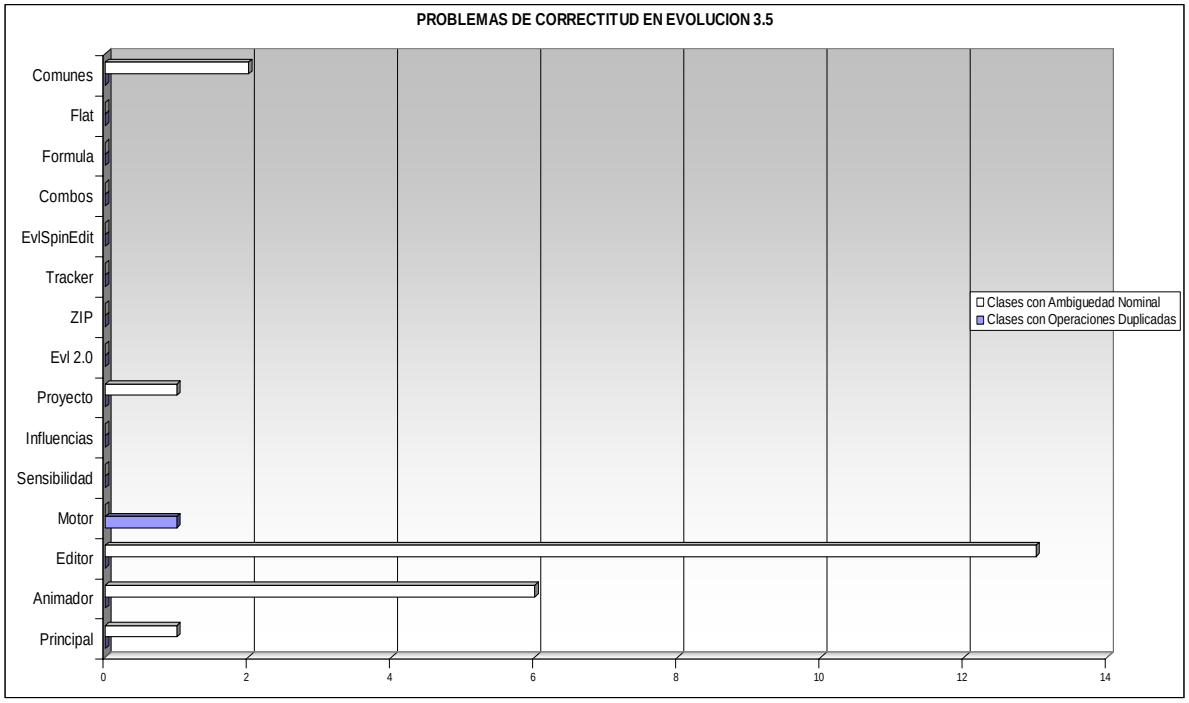

Figura 14: Distribución de los Problemas de CORRECTITUD x Módulo

d) Los datos de NO-Correctitud discriminados por cada módulo se presentan a continuación:

Módulo	Num. Violaciones a la CORRECTITUD
Principal	1
Animador	6
Editor	13
Motor	1
Sensibilidad	0
Influencias	0
Proyecto	1
EVL2.0	0
ZIP	0
Tracker	0
EVLSpinEdit	0
Combos	0
Formula	0
Flat	0
Comunes	2

Tabla 8: Distribución de Problemas de Completitud x Módulo

- **NOMBRADO y ESTILO.** En total 87 problemas encontrados.

a) El 8% de las clases tiene entre atributos y operaciones más de 60, en otras palabras son clases enormes. En total 35 de las 448 clases presentan esta violación a la regla. Estas clases usualmente se convierten en cuellos de botella para el mantenimiento, son causa de problemas de confiabilidad y signo de una débil concepción en la arquitectura orientada a objetos.

b) Mientras que el 2% de las clases padece problemas de ciclos de dependencia circular. En total 9 de las 448 clases presentan esta violación a la regla. Es posible encontrar serios inconvenientes a la hora de probar y re-usar de forma independiente estas clases. El problema se agrava si las clases provienen de distintos paquetes.

c) El 10% de las clases del sistema, sobre-escribe sus atributos heredados. En total 43 de las 448 clases presentan esta violación a la regla.

d) En la Figura 15 se ilustra la distribución de problemas encontrado para la categoría de NOMBRADO y ESTILO, discriminada por módulos. Allí puede observarse una marcada concentración de problemas en el componente FLAT (mayor cantidad de clases enormes), seguido del animador, influencias (mayor cantidad de clases con sobre-escritura de atributos heredados) y el componente ZIP.

e) Los datos de NOMBRADO y ESTILO discriminados por cada módulo se presentan a continuación:

Módulo	Num. Violaciones NOMBRADO y ESTILO
Principal	1
Animador	16
Editor	4
Motor	3
Sensibilidad	5
Influencias	13
Proyecto	2
EVL2.0	0
ZIP	10
Tracker	0
EVLSpinEdit	0
Combos	0
Formula	4
Flat	22
Comunes	7

Tabla 9: Distribución de Problemas de Nombrado y Estilo x Modulo

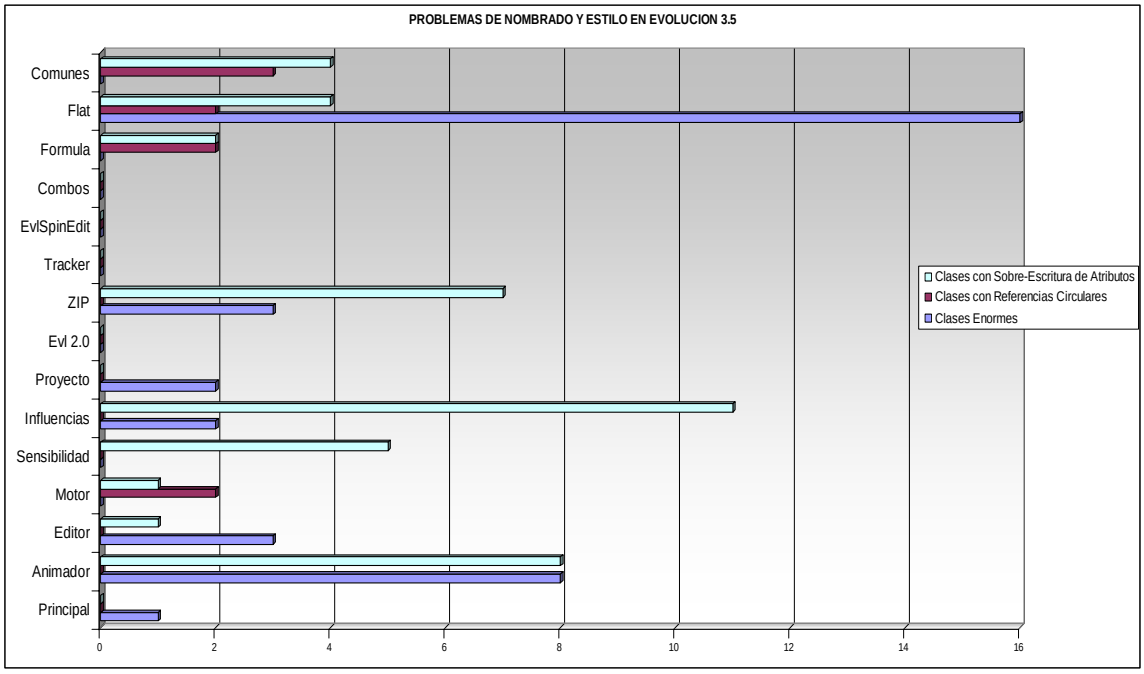

Figura 15: Distribución de Problemas de NOMBRADO y ESTILO x Módulo

- **ACOPLAMIENTO**

a) El factor de acoplamiento para las clases del sistema arrojó un valor del 2%. Es un acoplamiento bajo considerando que el máximo número de acoplamientos del sistema es TC^2-TC=200.256, donde TC=448 clases. Esto es posible al uso intensivo de componentes. Aún así, existen 3239 dependencias entre las clases del sistema, lo cuál es complejo de entender si el modelo está desordenado, incompleto y poco especificado.

b) En el mismo sentido, el factor de acoplamiento para paquetes del sistema arrojó un valor del 5%, considerado bajo dado que el número máximo de acoplamientos del sistema para los paquetes es TP^2-TP=50.882, donde TP=239. Sin embargo este dato es engañoso debido a que en la mayoría de los casos, un paquete está ocupado por solo una clase. Aún así existen 2562 dependencias entre paquetes, lo cuál se

torna complejo de entender cuando el modelo está pobremente descrito o desordenado.

- **HERENCIA**

  - De los 8611 atributos totales de las clases del sistema, 3516 son atributos heredados, arrojando un valor de herencia en atributos del 41%, lo cuál implica un uso intensivo de este mecanismo.

  - Asimismo, de las 8891 operaciones totales de las clases del sistema, 4168 corresponden a operaciones heredadas es decir se utiliza este mecanismo un 47% de las veces.

- **ENCAPSULAMIENTO**

  - De los 5095 atributos totales de las clases del sistema, 3203 son atributos públicos, es decir que sólo el 37% de los atributos del sistema se está ocultando.

  - Asimismo, de las 4723 operaciones totales de las clases del sistema, 2269 corresponden a operaciones públicas es decir se oculta el 52% de los métodos.

Por último, la Figura 16, ilustra el estado de los factores de Completitud, Correctitud, Nombrado, Estilo, Documentación, Acoplamiento, Herencia y Encapsulamiento para el sistema. Los datos mostrados son independientes unos de otros y deben tomarse en relación a su propia categoría.

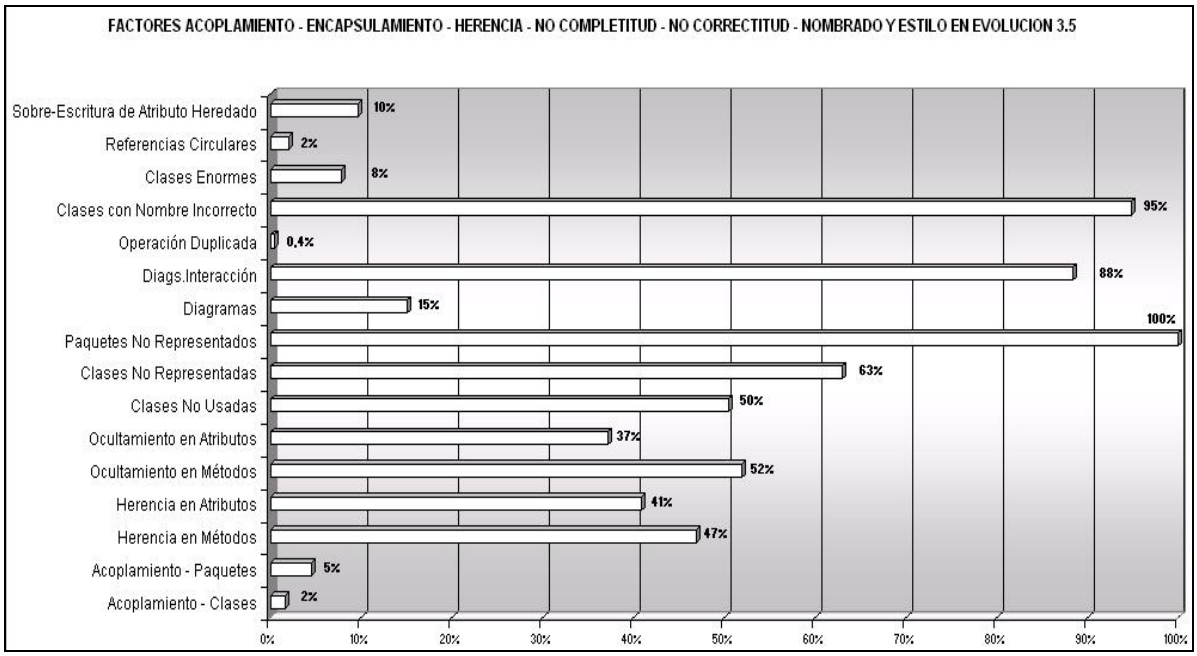

Figura 16: Estado final de los Atributos de la Arquitectura de EVOLUCION 3.5

4.1.9. Sugerencias para el Mejoramiento Arquitectónico

A continuación se describen los cambios que deben realizarse a la arquitectura de la herramienta EVOLUCION 3.5 con el propósito de mejorar el estado de sus atributos en aras de alcanzar una arquitectura más organizada, fácil de entender, mantener y extender en el marco de una comunidad (i+d). Con el objeto de facilitar la lectura, se organizarán las sugerencias en el mismo orden de aparición y categorías que los del numeral anterior.

- **COMPLETITUD.** En total 4081 problemas encontrados por resolver.

    a) **Sugerencia 1.** Representar en el modelo de la documentación oficial de EVOLUCION 3.5 el 100% de los paquetes encontrados en el código fuente. En total se resolverían 239 de los 239 problemas encontrados.

b) **Sugerencia 2.** Representar en el modelo que entregó SIMON el 63% de las clases encontradas en el código fuente que aún no están representadas. En total se resolverían 282 problemas.

c) **Sugerencia 3.** Completar el 85% de la documentación UML (casos de uso y diagramas de secuencia) del sistema, resolviendo al instante 158 problemas de especificación y completitud para estos artefactos. Además completar el modelo de interacción en un 88% eliminando 3176 problemas de completitud adicionales.

d) **Conclusión.** Al realizar las mejoras sugeridas se habrán resuelto un total de 3855 problemas de COMPLETITUD, mejorando notablemente la facilidad de comprensión de la arquitectura de EVOLUCION 3.5, por parte de una comunidad de desarrollo. A continuación se muestra el conteo de problemas de COMPLETITUD antes y después de las mejoras:

	COMPLETITUD											
	ANTES						DESPUES					
	Clases No Usadas	Clases No Representadas	Paquetes No Representados	Cusos No Especificados	Diags. Interacción	Total Problemas	Clases No Usadas	Clases No Representadas	Paquetes No Representados	Cusos No Especificados	Diags. Interacción	Total Problemas
PRINCIPAL	2	0	3	N.D	25	30	2	0	0	N.D	0	2
ANIMADOR	33	39	66	64	560	762	33	0	0	0	0	33
EDITOR	27	4	24	36	271	362	27	0	0	0	0	27
MOTOR	1	0	7	58	80	146	1	0	0	0	0	1
SENSIBILIDAD	21	26	22	N.D	208	277	21	0	0	N.D	0	21
INFLUENCIAS	12	34	13	N.D	272	331	12	0	0	N.D	0	12
PROYECTO	1	0	3	N.D	8	12	1	0	0	N.D	0	1
EVL2.0	1	0	1	N.D	8	10	1	0	0	N.D	0	1
ZIP	92	124	31	N.D	992	1239	92	0	0	N.D	0	92

	COMPLETITUD											
	ANTES						DESPUES					
	Clases No Usadas	Clases No Representadas	Paquetes No Representados	Cusos No Especificados	Diags. Interacción	Total Problemas	Clases No Usadas	Clases No Representadas	Paquetes No Representados	Cusos No Especificados	Diags. Interacción	Total Problemas
TRACKER	1	1	1	N.D	8	11	1	0	0	N.D	0	1
EVLSpinEdit	0	1	1	N.D	8	10	0	0	0	N.D	0	0
COMBOS	1	3	1	N.D	24	29	1	0	0	N.D	0	1
FORMULA	7	12	5	N.D	80	104	7	0	0	N.D	0	7
FLAT	14	35	30	N.D	280	359	14	0	0	N.D	0	14
COMUNES	13	3	31	N.D	352	399	13	0	0	N.D	0	13
Totales	**226**	**282**	**239**	**158**	**3176**	**4081**	**226**	**0**	**0**	**0**	**0**	**226**

Tabla 10: COMPLETITUD Antes y Después de las Mejoras

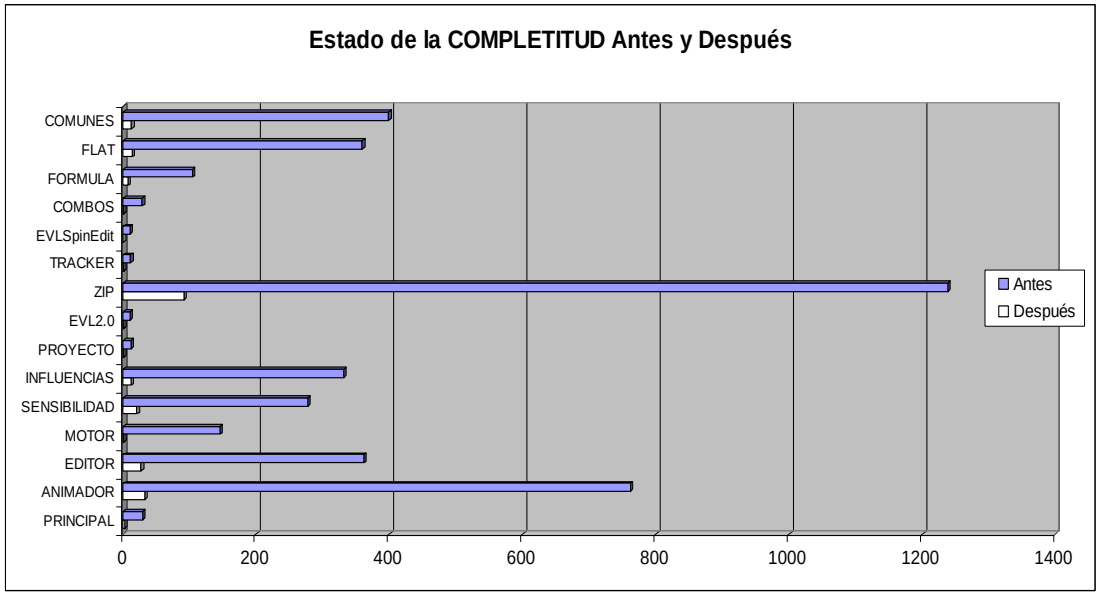

Figura 17: COMPLETITUD Antes y Después

- **CORRECTITUD.** En total 25 problemas encontrados por resolver.

  a) **Sugerencia 1.** Debe corregirse el 95% de los nombres de las clases del modelo para que se correspondan con las encontradas en el código fuente. En total se resuelven 23 de los 23 problemas encontrados en este aspecto.

  b) **Sugerencia 2.** Debe revisarse las operaciones duplicadas de la clase "**TManejadorEvaluacion**" y eliminar si es el caso, una de ellas. En total 2 errores se resolverían.

  c) **Conclusión.** Al realizar las mejoras sugeridas se habrán resuelto un total de 25 problemas de CORRECTITUD, aportando facilidad de comprensión y facilidad de mantenimiento debido a la coherencia entre el modelo y el código fuente. Sin embargo, los 2 problemas de operación duplicada no se resolverán, sólo se dejarán indicados. A continuación se muestra el conteo de problemas de CORRECTITUD antes y después de las mejoras:

	CORRECTITUD					
	ANTES			**DESPUES**		
	Operaciones Duplicadas	**Nombre Incorrecto**	**Total Problemas**	**Operaciones Duplicadas**	**Nombre Incorrecto**	**Total Problemas**
PRINCIPAL	0	1	1	0	0	0
ANIMADOR	0	6	6	0	0	0
EDITOR	0	13	13	0	0	0
MOTOR	2	0	2	2	0	2
SENSIBILIDAD	0	0	0	0	0	0
INFLUENCIAS	0	0	0	0	0	0
PROYECTO	0	1	1	0	0	0

	CORRECTITUD					
	ANTES			DESPUES		
	Operaciones Duplicadas	Nombre Incorrecto	Total Problemas	Operaciones Duplicadas	Nombre Incorrecto	Total Problemas
EVL2.0	0	0	0	0	0	0
ZIP	0	0	0	0	0	0
TRACKER	0	0	0	0	0	0
EVLSpinEdit	0	0	0	0	0	0
COMBOS	0	0	0	0	0	0
FORMULA	0	0	0	0	0	0
FLAT	0	0	0	0	0	0
COMUNES	0	2	2	0	0	0
Totales	**2**	**23**	**25**	**2**	**0**	**2**

Tabla 11: Estado de la CORRECTITUD antes y Después de las Mejoras

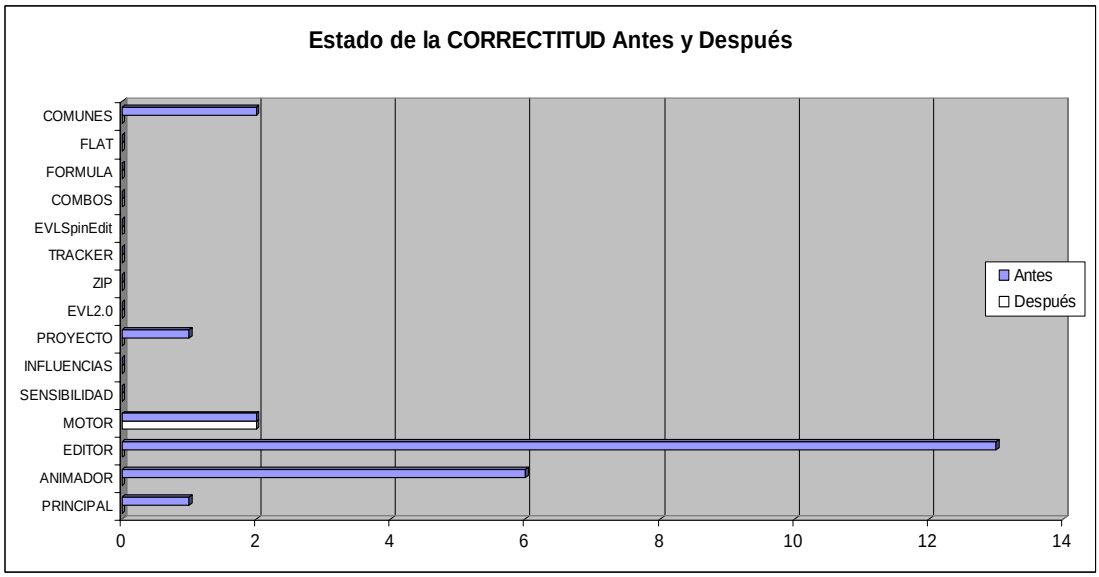

Figura 18: CORRECTITUD antes y después de las mejoras

- **NOMBRADO y ESTILO.** En total 87 problemas encontrados por resolver.

a) **Sugerencia 1.** En total 35 (8%) de las 448 clases son clases enormes. Debe considerarse su distribución en clases más pequeñas. Si no existe justificación para esta violación entonces se deben resolver estos 35 problemas.

b) **Sugerencia 2.** Las clases: **TFunctionList**, **TFormulaLib**, **TFlatAnimWnd**, **TFlatAnimHookWnd**, **TListaElementosEvaluacion**, **TListaConecciones**, **TLineaDinamico**, **TElementoEvaluacion** y **TElemento,** tienen problemas de referencias circulares y se recomienda eliminarlas. Si no existe justificación para esta violación se debe proceder a resolver estos 9 problemas.

c) **Sugerencia 3.** En total 43 de las 448 clases sobre-escribe sus atributos heredados. Debe considerarse cambiar el nombre de estos atributos, si no existe justificación para esta violación. Se resolverían 258 problemas.

f) **Conclusión.** Al realizar las mejoras sugeridas se resolverían 87 problemas de NOMBRADO y ESTILO. Sin embargo debe advertirse que emprender este esfuerzo es bastante delicado y propenso a errores, por lo tanto se sugiere realizar estos cambios en presencia de los desarrolladores de EVOLUCION 3.5. En la siguiente tabla se especifica cómo quedó el estado de esta categoría de NOMBRADO y ESTILO en el contexto de la nueva arquitectura.

En caso de que el lector del presente documento se interese en alcanzar un mayor nivel de detalle acerca de los datos recaudados a partir de la evaluación arquitectónica de EVOLUCION, se recomienda ver el Anexo 16.

	NOMBRADO y ESTILO							
	ANTES				DESPUES			
	Clases Enormes	Clases con Referencias Circulares	Clases que Sobre-Escriben Atributos Heredados	Total Problemas	Clases Enormes	Clases con Referencias Circulares	Clases que Sobre-Escriben Atributos Heredados	Total Problemas
PRINCIPAL	1	0	0	1	1	0	0	1
ANIMADOR	8	0	8	16	8	0	8	16
EDITOR	3	0	1	4	3	0	1	4
MOTOR	0	2	1	3	0	2	1	3
SENSIBILIDAD	0	0	5	5	0	0	5	5
INFLUENCIAS	2	0	11	13	2	0	11	13
PROYECTO	2	0	0	2	2	0	0	2
EVL2.0	0	0	0	0	0	0	0	0
ZIP	3	0	7	10	3	0	7	10
TRACKER	0	0	0	0	0	0	0	0
EVLSpinEdit	0	0	0	0	0	0	0	0
COMBOS	0	0	0	0	0	0	0	0
FORMULA	0	2	2	4	0	2	2	4
FLAT	16	2	4	22	16	2	4	22
COMUNES	0	3	4	7	0	3	4	7
Totales	**35**	**9**	**43**	**87**	**35**	**9**	**43**	**87**

Tabla 12: Estado de NOMBRADO y ESTILO Antes y Después de las Mejoras

Finalmente, con el propósito de facilitar la lectura de este documento en lo relacionado con la interpretación de los resultados de la evaluación arquitectónica y las sugerencias a la nueva arquitectura, se ha dispuesto el Anexo 17, el cuál presenta de manera extensa estos aspectos. A continuación se presenta brevemente el proceso iterativo e incremental que facilitó la reconstrucción arquitectónica de EVOLUCION.

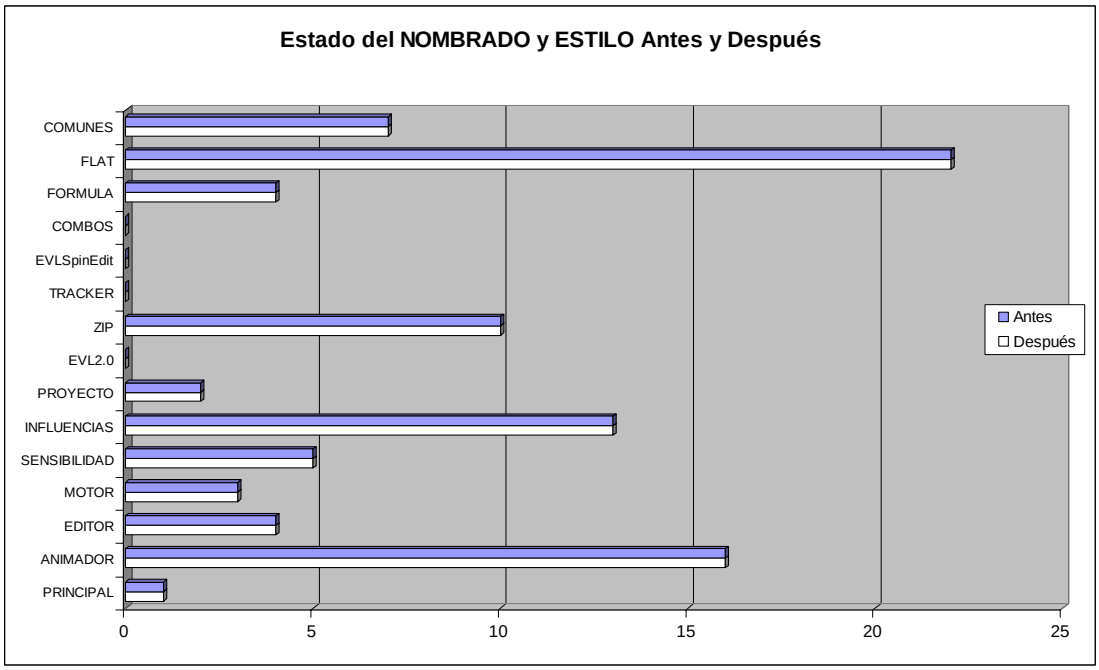

Figura 19: NOMBRADO y ESTILO Antes y Después

4.1.10. Presentación de la Nueva Arquitectura

La reconstrucción arquitectónica es un proceso interpretativo, interactivo e iterativo, y no un proceso automático. Este requiere la habilidad y atención tanto de expertos en ingeniería inversa como de los arquitectos de software. La reconstrucción arquitectónica busca generar una representación de la arquitectura del sistema que puede ser usada de diversas formas. Su principal uso consiste en documentar la arquitectura implementada de un sistema. Si la documentación existente o disponible es obsoleta, la recuperación de la arquitectura puede ser utilizada como base para su re-documentación. La representación de la arquitectura también puede ser usada como punto de partida para la reingeniería del sistema. Por último, esta representación puede ser útil para identificar la funcionalidad de los componentes para reutilizarlos o establecerlos como la arquitectura base.

A continuación se describe el camino recorrido para la concepción de la nueva arquitectura de EVOLUCION 3.5. Esta narración se distribuye en cuatro (4) iteraciones.

Iteración 0.

- **Descripción.** Durante esta iteración, se importó el archivo Modelo_Evolucion_Viejo_Generado.xmi, cuyo formato XMI, es aceptado por la herramienta Sybase PowerDesigner, la cuál generó la totalidad de las clases halladas en el archivo, sin embargo las unidades o paquetes, no fueron generados, tampoco fueron generados los atributos cuyo tipo de datos es otra clase. El estado inicial de esta reconstrucción arquitectónica se puede constatar en la Figura 20 y/o también en el archivo Evolucion_Iteracion_0.oom. Allí se puede apreciar el estado de alta entropía alcanzado en lo relativo a la conformidad del modelo entregado por SIMON y el código fuente de EVOLUCION 3.5.

Iteración 1.

- **Descripción.** Durante esta iteración, se crearon todos los paquetes lógicos que se corresponden con las unidades escritas en Delphi 7.0. Con el propósito de mantener y respetar el espíritu de los desarrolladores de EVOLUCION 3.5, al mismo tiempo que se facilita el entendimiento de la arquitectura, los paquetes lógicos creados, reflejan la estructura de directorios que almacenan los archivos .PAS de la carpeta "Source_Code".

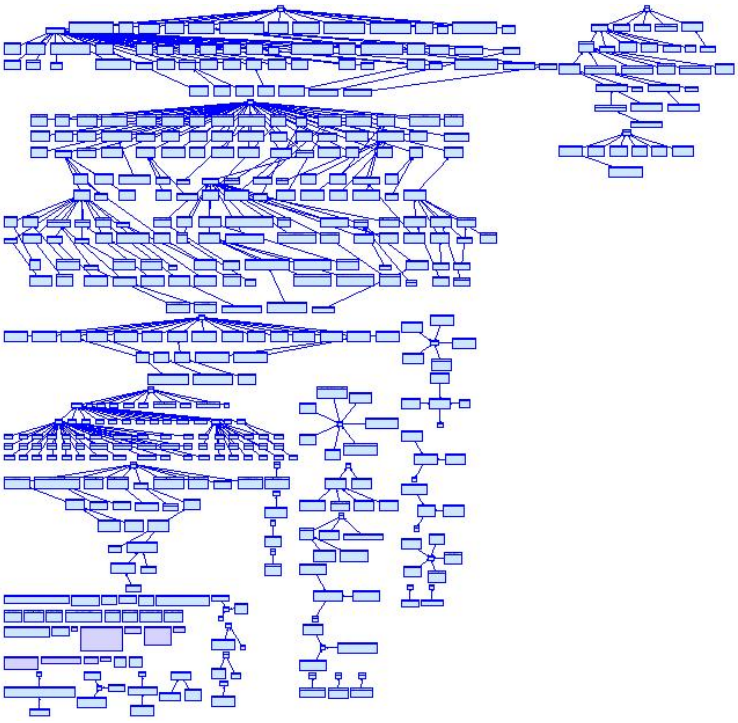

Figura 20: Visión de la Arquitectura de EVOLUCION en la Iteración 0

Iteración 2.

- **Descripción.** Durante esta iteración, se distribuyeron todas las clases generadas durante la ***iteración 0*** en los paquetes de la ***iteración 1*** según la herramienta Ess-Model.

Iteración 3.

- **Descripción.** Durante esta iteración se crearon tanto los diagramas de clase como los diagramas de paquetes de toda la herramienta EVOLUCION 3.5. Debe enfatizarse que esta fue la más compleja e intensa de todas las iteraciones de la reconstrucción arquitectónica. Nuevamente, fue de gran ayuda la herramienta Ess-Model. La siguiente figura muestra el estado de organización alcanzado al final de esta iteración para el diagrama de paquetes principal.

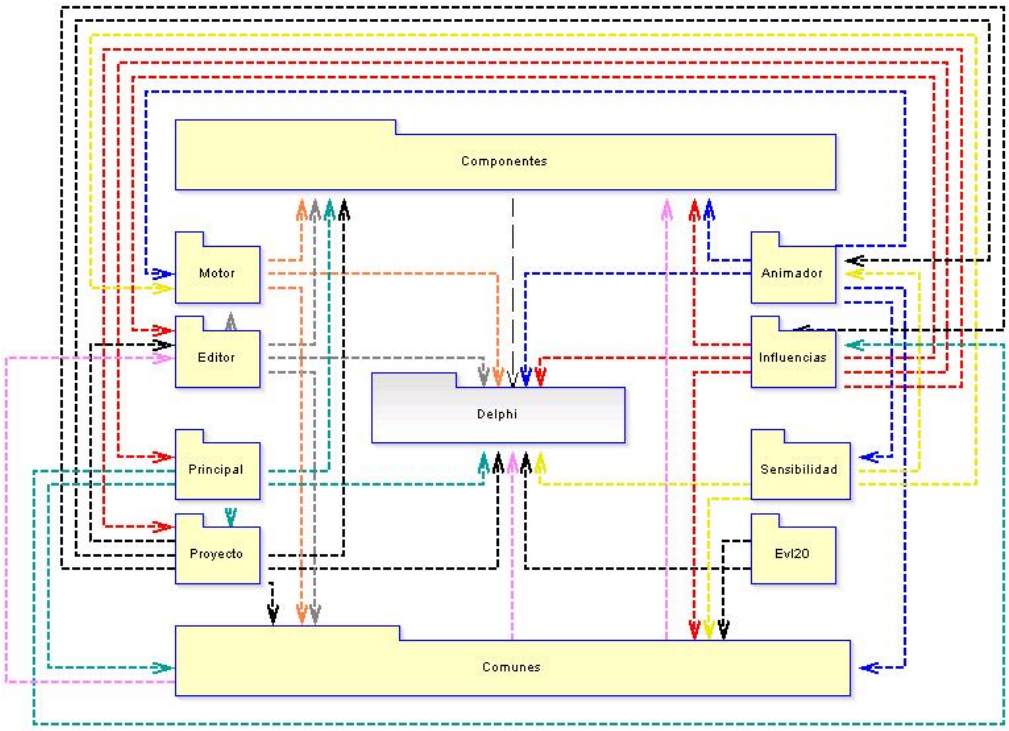

Figura 21: Visión de la Arquitectura de EVOLUCION en la Iteración 3

Iteración 4.

Descripción. Durante esta iteración se crearon los diagramas de casos de uso así como su especificación de alto nivel. Para tal propósito se uso la herramienta EVOLUCION 3.5 y el IDE de Borland Delphi 7.0.

Con el propósito de facilitar la lectura de este documento al mismo tiempo de presentar en un formato más adecuado la arquitectura de EVOLUCION, se ha dispuesto el Anexo 2, el cuál consiste en un archivo de la herramienta PowerDesigner, este último compila en un solo documento el modelo de casos de uso, los diagramas de clase, los diagramas de paquetes, el modelo de componentes y la especificación de los requisitos funcionales y no funcionales del ESMS-MI.

5. PROCESO DE SOFTWARE BAJO UN AMBIENTE DISTRIBUIDO

5.1. INTRODUCCIÓN

Este capitulo se concentra en la satisfacción del primer objetivo general de este trabajo, el cual propone:

- *"Elaborar una instanciación de un proceso adecuado para el desarrollo distribuido de software que favorezca las condiciones mínimas para unificar e integrar la comunicación y gestión durante la ejecución de un proyecto software orientado a entregar un "Entorno de Modelamiento-simulación de modelos integrados", liderado por el grupo SIMON."*

La organización de un proceso de desarrollo de software, es una actividad compleja que abarca la gestión de aspectos tales como: tareas, recursos, tiempo, productos, personal, tecnología, que son utilizados, consumidos o producidos durante el ciclo de vida del proyecto. De otra parte, la organización de un proceso debe involucrar la definición de la metodología y las herramientas CASE que la soportan. Según Carvalho & Chiossi (2001), *"una metodología de desarrollo de software, detalla las actividades del ciclo de vida, especificando un conjunto único y coherente de principios, métodos, lenguaje de representación, normas, procedimientos e instrumentación, los cuáles permiten al desarrollador de software, implementar las especificaciones derivadas de cada fase del ciclo de vida del software sin ambigüedad".*

En consecuencia, este capítulo, se concentra en la especificación de un proceso de desarrollo de software basándose en metodologías ya existentes mediante una selección, adecuación y armonización de aspectos clave para concebir un marco idóneo para el desarrollo ágil de software bajo un ambiente de dispersión geográfica. Este proceso de desarrollo se denominará de ahora en adelante "**AGILE-DISOP**", acrónimo de proceso ágil de desarrollo distribuido de software.

Para facilitar la lectura, este capítulo especificará mediante una secuencia lógica los elementos que van dando forma a la metodología **AGILE-DISOP**. En primer lugar, los elementos más tempranos corresponden a desafíos, requisitos y prácticas que **AGILE-DISOP** debe enfrentar, satisfacer y promover respectivamente. En segundo lugar, los elementos intermedios corresponden a la especificación de los diversos sub-modelos (denominados disciplinas) que integran **AGILE-DISOP**, los cuáles son explicados mediante una serie de descriptores como se ilustra en la Figura 24: Visión General de Agile-DISOP. Y finalmente, se presentan elementos correspondientes a la instrumentación y soporte del **AGILE-DISOP**, representados por artefactos (plantillas de documentos entre otros) y la plataforma seleccionada para integrar a la comunidad de desarrollo, la cuál ha pasado por un proceso de búsqueda, recopilación, evaluación y selección que también se describe en este capítulo.

5.2. DESAFIOS DE AGILE-DISOP

En el capítulo 3, numeral 3.1.1, se enunciaron algunas peculiaridades relacionadas con los desafíos con los que debía enfrentarse el desarrollo de software bajo ambientes de dispersión geográfica. A modo de recuerdo, se presenta una síntesis de aquellos desafíos antes citados, pero complementados con otros, que son producto de una investigación más profunda.

DESAFÍO	**CUESTIONES IMPORTANTES A RESOLVER**
Comunicación	a. Cómo se relacionan las personas con otras a través de las TIC? b. Cómo reaccionan las personas presencial y/o virtualmente? c. Cómo interpretan las personas las reacciones en otras?
Coordinación	a. Cómo sincronizar la disponibilidad de las personas? b. Cómo coordinar e integrar las actividades? c. Cómo compartir datos, información y aplicaciones?

DESAFÍO	CUESTIONES IMPORTANTES A RESOLVER
Infraestructura	a. Que criterios determinan la selección adecuada del Hardware y el Software que soportarán el proceso bajo un ambiente distribuido? b. Como soportar eficazmente las áreas de gestión, comunicación, coordinación y formalización mediante servicios informáticos disponibles a toda hora, en todo lugar y para todos los miembros?
Gestión	a. Como implementar eficazmente el control sobre el avance y presupuesto del macro-proyecto y los sub-proyectos? b. Como implementar eficazmente la gestión de los riesgos del macro-proyecto y sub-proyectos? c. Como implementar un mecanismo eficaz para la asignación y evaluación de tareas y sus resultados? d. Como ejercer un liderazgo efectivo aún con miembros del equipo remotos favoreciendo un clima de confianza? e. Que estrategias deben considerarse para establecer un clima de pertenencia, confianza y solidaridad dentro de la comunidad de desarrollo?
Formalización	a. Como unificar un modelo de desarrollo a través de la comunidad? b. Como alcanzar compatibilidad en los diferentes estilos de desarrollo que usa la comunidad? c. Como alcanzar compatibilidad total de la documentación? d. Como integrar una suite de herramientas de soporte para la comunidad?

Tabla 13: Desafíos de Agile-DISOP

5.3. REQUISITOS DE AGILE-DISOP

En este apartado se enuncian los principales requisitos que debería satisfacer la metodología **Agile-DISOP** que se propone. Estos requisitos han sido determinados a partir de los desafíos encontrados para el desarrollo de software bajo un ambiente de dispersión geográfica. En primer lugar, **Agile-DISOP** adaptaría principios y prácticas provenientes de otros paradigmas que han demostrado ser exitosos y ampliamente utilizados, con el objeto de aprovechar todo el conocimiento, experiencia y tecnología sin incurrir en esfuerzos redundantes y aislados. Al nivel más alto, **Agile-DISOP** debería cumplir con los siguientes requisitos:

AREA	REQUISITOS
1. General	a. Debe ser efectivo y eficaz favoreciendo tiempos de desarrollo más cortos, buen retorno a la inversión y finalmente la calidad del producto final. b. Su adopción debe ser incremental. Al principio con poco esfuerzo y progresivamente extendiéndose hasta su implementación completa. c. Debe proveer resultados rápidos y tangibles con el objeto de que su uso y adopción continua se justifique. d. Debe adecuar y armonizar principios, prácticas, métodos y herramientas de demostrada eficacia. e. Debe proveer mecanismos para el mejoramiento continuo.
2. Comunicación	a. Debe concebir actividades periódicas y presenciales de todo el equipo de desarrollo. b. Debe concebir actividades periódicas y virtuales de todo el equipo de desarrollo y a través de todos los sub-equipos geográficamente dispersos. c. Debe concebir mecanismos eficaces para compartir documentos e información, así como consultas entre colegas.
3. Coordinación	a. Debe concebir mecanismos para la publicación, intercambio, ajuste y sincronización de las agendas personales o grupales con el propósito de coordinar todos los equipos distribuidos geográficamente?
4. Infraestructura	El software seleccionado como plataforma debe: a. Ser de fácil uso, interoperable con otras herramientas y portable a varias plataformas. b. Permitir la planificación de proyectos, asignación de recursos, presupuesto, tiempo y personal a tareas. c. Permitir el intercambio electrónico de información a través de todos los equipos dispersos geográficamente. d. Estar disponible, abierto y sin costo para todos los miembros del equipo geográficamente disperso. Se recomienda Internet. e. Estar en capacidad de reflejar una estructura organizacional de Macro-proyectos y sub-proyectos que incluya: o Agendas personales, grupales y globales. o Asignación de tareas personales, grupales y globales o Organización de contactos y reuniones. o Mensajería a través de e-mail, Chat y mensajes

AREA	REQUISITOS
	instantáneos o SMS. o Registro del esfuerzo personal y presentación de resultados consolidados por grupos y por toda la comunidad. o Organización de los miembros del equipo en perfiles con diversidad de derechos de acceso. o Organización, publicación y descarga de archivos. o Control de versiones en archivos y código fuente.
5. Gestión	a. Debe sugerir mecanismos eficaces para medir y/o controlar el avance del proyecto y su presupuesto. b. Debe sugerir mecanismos que implementan eficazmente la gestión de los riesgos del proyecto y sub-proyectos. c. Debe garantizar eficazmente la asignación y evaluación de tareas y sus resultados. d. Debe sugerir patrones para el liderazgo efectivo aún con miembros del equipo remotos. e. Debe sugerir patrones para establecer vínculos personales de confianza con miembros del equipo remotos. f. Debe sugerir estrategias para establecer un clima de pertenencia, confianza y solidaridad dentro de la comunidad de desarrollo.
6. Formalización	a. Debe unificar un estilo de desarrollo para toda la comunidad. b. Debe unificar el modelo de documentación para toda la comunidad. c. Debe unificar las herramientas de soporte y su entorno garantizando su disponibilidad, libre acceso y bajo costo para toda la comunidad. d. Llenar los vacíos dejados por las metodologías de desarrollo horizontales, las cuáles especifican como debe dirigirse el desarrollo de software de forma transversal indicando a veces lo “QUE” debe hacerse y omitiendo el “COMO” debe hacerse.

Tabla 14: Requisitos de Agile-DISOP

5.4. PRINCIPIOS Y BUENAS PRÁCTICAS DE AGILE-DISOP

En esta sección se han recopilado, seleccionado, adecuado y armonizado buenas prácticas que han demostrado su eficacia en el desarrollo de productos de software en diversidad de proyectos. Estas prácticas han sido tomadas desde modelos como

el Rational Unified Process (RUP 2003) hasta modelos de desarrollo de software libre como el Bazar, pero colocándolas en el marco de un desarrollo en comunidad concreto[59]. Las siguientes secciones, sintetizan el propósito de las prácticas propuestas por **Agile-DISOP.**

Nota: Cada numeral está escrito en un lenguaje "recomendativo" dirigido al propio usuario del proceso **Agile-DISOP.** Así mismo, todos los términos que requieren una interpretación especial, han sido indicados en letra cursiva dentro del cuerpo de documento y además han sido definidos en el Anexo 3.

5.4.1. Encárguese de las cuestiones de alto riesgo primero

Promueva una mitigación temprana de riesgos para todas aquellas características del sistema que son críticas, difíciles o arriesgadas en las *iteraciones* más tempranas, dejando el trabajo más fácil para las *iteraciones* tardías. Al mitigar los riesgos altos desde el principio, evitará que el proyecto "fracase tarde".

5.4.2. Estabilice una arquitectura cohesiva primero

Esta práctica se relaciona muy estrechamente con la planteada en el numeral 5.4.1, debido a que cuando se fomenta un enfoque de *"lo arriesgado va primero",* los aspectos arquitectónicos más importantes se van estabilizando por inercia. Promueva una implementación de la *arquitectura "en anchura y superficial",* es decir, estableciendo las decisiones de diseño importantes y los subsistemas con sus interfaces y responsabilidades durante las primeras iteraciones. Además, motive a la *comunidad de desarrollo*, a investigar en profundidad, en aquellas situaciones donde sean detectados *requisitos* difíciles o arriesgados.

---

[59] Es decir en el contexto de la comunidad de dinámica de sistemas y pensamiento sistémico que agrega a varias universidades publicas del territorio nacional.

5.4.3. Promueva el desarrollo Ágil, Iterativo, Incremental y Comunitario

Desarrolle iterativa e incrementalmente entregando una primera solución parcial tan pronto como sea posible, no importa que sea burda en principio, pero al menos debe convencer a la comunidad de que tiene el potencial de convertirse en algo útil en el futuro. Acepte el hecho de que sus primeras soluciones parciales pueden retroceder debido a rediseños que serán más estables con el tiempo. Involucre muy fuertemente a los miembros de la comunidad, tratándolos como co-desarrolladores, valorando sus ideas y aportes, sobre todo promoviendo un ambiente favorable para el co-desarrollo (debidamente gestionado con un sistema de control de versiones) y la co-depuración. Esto es mejor conocido como una estrategia de reclutamiento-retribución, bajo un enfoque de programación sin ego, mejor explicado en el numeral 5.4.10. Promueva una filosofía de desarrollo orientado hacia la reutilización permanente de código y/o componentes. Realice entregas de sus pequeños incrementos de software de manera periódica y frecuente y aproveche la sinergia de la comunidad para maximizar el esfuerzo dedicado al co-desarrollo y la co-depuración de estas entregas.

5.4.4. Promueva una gestión de requerimientos en Comunidad

Conciba mecanismos efectivos para encontrar, documentar, organizar y controlar en comunidad los siempre cambiantes *requerimientos* de un sistema. Promueva un ambiente de discusión sobre todos los requerimientos del sistema, especialmente sobre aquellos más inestables y no subestime los aportes de ningún miembro de la comunidad, antes bien, reconózcalos públicamente y en forma positiva, brindando mayor despliegue según la importancia del aporte mismo. Esto favorecerá una motivación individual en cada miembro de la comunidad basada en el reconocimiento público y auto estima personal, la cuál suele ser mucho más poderosa que las motivaciones económicas. Establezca un acuerdo global, no ambiguo y público sobre cada *requerimiento* del sistema y permita que sea expuesto a toda la comunidad, aproveche la sinergia de la crítica comunitaria para estabilizar

más rápidamente la definición óptima del requerimiento. Conciba mecanismos para determinar que dependencias entre *requerimientos* son importantes de monitorear y controlar. Y finalmente, conforme un *equipo* líder para la unificación y estabilización de los *requerimientos* ante la *comunidad de desarrollo*. Sobre los *requerimientos* debe tenerse muy en claro lo siguiente para no correr riesgos:

- **No siempre son obvios y pueden provenir de diversas fuentes. No siempre están claramente expresados.** Tenga en cuenta de que no importa que tan cuidadosamente se han definido los *requerimientos*, estos siempre cambiarán, causando un impacto no solo en las características del sistema que los implementan, sino también en las características del sistema relativas a *requerimientos* relacionados.

- **Los requerimientos pueden estar organizados en diversas categorías y niveles.** No necesariamente todos tienen la misma importancia. Lo usual es que se relacionen entre si y también con otros sub-productos (*artefactos*) del proyecto. Finalmente tenga en cuenta de que el número de *requerimientos* puede volverse inmanejable si no se controla.

5.4.5. Promueva una arquitectura basada en Componentes

El desarrollo basado en componentes es una variación del desarrollo tradicional de aplicaciones de software. Este tipo de desarrollo es más adecuado para particionar los esfuerzos de programación sin conflictos bajo ambientes de dispersión geográfica. Con este enfoque, la aplicación se construye mediante la implementación de componentes ejecutables desarrollados de forma independiente por equipos distintos. Estos componentes deben conformarse a interfaces bien definidas y consensuadas en comunidad. No olvide el uso de *patrones de diseño* para concebir un diseño elegante, sencillo y funcional. De otra parte, incremente su aplicación mediante la actualización y/o mejora de algunos de sus componentes. Considere la posibilidad de compartir componentes entre aplicaciones creando

oportunidades para la reutilización, pero tenga en cuenta la generación de inter-dependencias entre proyectos. Usando un estilo de desarrollo como el sugerido en el numeral 5.4.3, identifique que componentes debería desarrollar (secuencial o concurrentemente), reutilizar o comprar. Considere distintos grados de *granularidad* para sus pruebas, utilice pruebas de unidad para componentes individuales, pruebas de integración para el componente versus su entorno y pruebas globales para el sistema versus todos los componentes recién actualizados o integrados.

5.4.6. Promueva el modelado visual del sistema

Fomente una conciencia comunitaria que sea positiva respecto al modelamiento visual del sistema. Promueva una sensibilización respecto a las ventajas derivadas de esta práctica las cuáles son:

- **Mejora sobre el entendimiento de sistemas complejos.** Fomente la construcción de modelos para que todos los co-desarrolladores de la comunidad se mantengan enfocados en los aspectos esenciales de la *arquitectura* del sistema (componentes y sus interacciones). El modelado ayudará a la comunidad a visualizar, construir y documentar la *arquitectura* y comportamiento del sistema al mismo tiempo que se maneja la complejidad.

- **Exploración y evaluación de alternativas de diseño a bajo costo.** Convenza a su comunidad de utilizar la creación y/o modificación de modelos como una alternativa de exploración de bajo costo para el diseño. Aquellas ideas innovadoras pueden capturarse y revisarse por co-desarrolladores antes de invertir tiempo, dinero y esfuerzo en desarrollar código. Acoplado con un estilo de desarrollo como el propuesto en el numeral 5.4.3, el modelado visual, ayudará a la comunidad a evaluar y propagar los cambios en el diseño de forma rápida y económica.

- **Implementación sin improvisación**. Promueva el uso efectivo de las herramientas CASE disponibles como un soporte para la transformación de

modelos del diseño en código de base para la implementación, esto se conoce mejor como "forward engineering" o "generación de código". La ingeniería inversa o “reverse engineering” puede ser aplicada para generar modelos de diseño a partir de código existente, pudiendo ser útil para la evaluación. También puede usar "Round trip engineering" o “Ingeniería de Viaje Redondo” la cual combina generación de código e ingeniería inversa para asegurar la consistencia y sincronización entre el diseño y el código.

- **Captura precisa de requerimientos.** Tenga en cuenta que los requerimientos expresados en forma precisa y no-ambigua ayudarán más efectivamente a todos los interesados en el proyecto, a entender el problema y ponerse de acuerdo sobre su propósito. La expresión clara de los requerimientos separa el comportamiento externo del sistema de los detalles de su implementación, ayudando a concentrarse más en el propósito mismo del sistema que en problemas de programación.

- **Comunicación de aspectos clave del diseño sin ambigüedad.** Unifique el uso de un lenguaje de modelamiento (Ej.: UML), el cuál le permitirá comunicar sin ambigüedades las decisiones de diseño que no son obvias y que no pueden inferirse a partir del código. Además su utilización le permitirá el uso de una semántica rica y suficiente para capturar todas las decisiones tácticas y estratégicas. Brinde a los miembros de su comunidad la posibilidad de razonar sobre la arquitectura del sistema, superando las barreras del lenguaje y promueva el uso de herramientas CASE capaces de interpretar y traducir a código los modelos UML.

5.4.7. Verifique calidad constantemente aprovechando la comunidad

Promueva un espíritu comunitario en la verificación constante y creciente de la calidad. Evalúe en comunidad la calidad de todos los *artefactos* y las *actividades* que los producen al final de cada iteración. Cree diversos escenarios de

demostración y prueba para el código ejecutable y publique los resultados a toda la comunidad para exponerlos a debate, así, más ágilmente se negociarán acuerdos sobre los *requerimientos* y el diseño del sistema. Aproveche la sinergia de la comunidad para resolver problemas. Muchas cabezas sobre un problema determinado encontrarán una solución más rápidamente. Así mismo, promueva un espíritu de calidad comunal, cuyo origen es el co-desarrollador mismo, el cuál mediante su aporte comprometido y desinteresado, propaga los beneficios derivados de la elaboración de sus *artefactos* y *actividades* hacia toda la comunidad. Defina un conjunto coherente y práctico de *métricas* para valorar la calidad tanto del proceso como del producto, y utilícelo en el marco de un modelo de mejoramiento para el proceso de desarrollo o SPI por ejemplo IMPACT, descrito en el numeral 3.3.

5.4.8. Promueva la Gestión del Cambio en Comunidad

Una característica importante del desarrollo en comunidad consiste en disponer de una gran cantidad de desarrolladores organizados en múltiples *equipos* dispersos geográficamente, todos trabajando juntos en múltiples *iteraciones*, versiones del producto software y herramientas de desarrollo o soporte, pero sobre una plataforma que permite su integración como comunidad. En estas condiciones sin un mecanismo de monitoreo y control, el proceso de desarrollo puede degenerar rápidamente en un caos. Cree mecanismos que permitan implementar informaticamente la *trazabilidad* entre todos los *artefactos* en cualquier iteración (para cualquier sub-proyecto), de tal forma que todo cambio en un *artefacto* debería propagarse a través de todos sus *artefactos* relacionados. Así mismo, emplee un sistema de control de versiones para el código desarrollado por la comunidad (se recomienda CVS) e intégrelo a su plataforma de gestión de proyecto en comunidad.

5.4.9. Promueva el uso de una plataforma de integración en comunidad

Aproveche las TIC´s[60], como mecanismo de integración para la comunidad, hoy en día existen soluciones de código abierto, disponibles para las comunidades de desarrollo. Tenga en cuenta de que es supremamente importante de que cada miembro de la comunidad pueda sentirse parte de ella desde cualquier lugar y a cualquier hora. La plataforma seleccionada debe ofrecer servicios para la integración efectiva de la gestión, comunicación y coordinación de los equipos geográficamente dispersos. Sin embargo, es recomendable contar con servicios automatizados para el control de las versiones de código, como se menciono en el apartado anterior. En el numeral 5.15 se describe el proceso de búsqueda y selección de una plataforma Web que integraría a la comunidad de dinámica de sistemas en el marco del desarrollo del ESMS-MI.

5.4.10. Trate a los usuarios como co-desarrolladores

Tratar a los usuarios como co-desarrolladores (colaboradores) es la forma más apropiada de mejorar el código, y la más efectiva de depurarlo. Promoviendo la práctica propuesta en el numeral 5.4.3 (liberar más rápido y con más frecuencia), puede aprovechar la sinergia de la comunidad para escuchar que defectos o mejoras podrían integrarse a los incrementos publicados. Al respecto es recomendable una filosofía de estímulo y reconocimiento constante para aquellos usuarios que toman parte en la acción. Cada colaborador es premiado mediante el reconocimiento y exhibición (a menudo diaria) de sus aportes, de esta forma la posesión intelectual y egoísta de cada desarrollador sobre su código desaparece y es reemplazada por un ánimo constante por colaborar para ser reconocido públicamente ante toda la comunidad, esto es mejor conocido como programación sin ego. La idea detrás de esta estrategia, consiste en maximizar la cantidad de

[60] Tecnologías de la información y las comunicaciones

horas-hombre dedicadas a la depuración y desarrollo del proyecto en comunidad. Con el tiempo y dada una base suficiente de desarrolladores, colaboradores y personal de prueba, cualquier problema puede ser caracterizado rápidamente y su solución será obvia al menos para alguien dentro de la comunidad.

- **Reflexión.** Aquí, yace una diferencia esencial entre dos estilos. En el enfoque usado para desarrollar software propietario, los errores y problemas de desarrollo son fenómenos complejos y profundos. Generalmente toma meses de revisión exhaustiva para que unos cuantos alcancen la seguridad de que los errores han sido eliminados del todo. Por eso se dan los intervalos tan largos entre cada versión que se libera y la inevitable desmoralización cuando estas versiones, largamente esperadas, no resultan perfectas. En el enfoque usado para desarrollar software abierto y en comunidad, se asume que los errores son fenómenos relativamente evidentes, cuando se exhiben a miles de entusiastas desarrolladores que colaboran sobre cada una de las versiones. Por lo tanto, se libera con frecuencia para poder obtener una mayor cantidad de correcciones en menos tiempo. Una mayor cantidad de usuarios detecta más errores debido a que tienen diferentes maneras de evaluar el programa. Este efecto se incrementa cuando los usuarios son desarrolladores asistentes. Cada uno enfoca la tarea de la caracterización de los errores con instrumentos analíticos distintos, desde un ángulo diferente.

Promueva un espíritu de "*tratar a los colaboradores como si fueran el recurso más valioso",* así pues, ellos responderán convirtiéndose en el recurso más valioso. Un coordinador o líder de proyecto debe tener una excelente capacidad de comunicación. Para poder construir una comunidad de desarrollo se necesita atraer gente, interesarla en lo que se está haciendo y mantenerla contenta con el trabajo que se está desarrollando. El entusiasmo técnico constituye una buena parte para poder lograr esto, pero está muy lejos de ser definitivo. Además, es importante la personalidad que se proyecta. Por tanto, lo más importante, después de tener

buenas ideas, es reconocer las buenas ideas de los usuarios. Esto último es a veces mejor. En consecuencia, no es crítico que el coordinador del proyecto sea capaz de originar diseños de calidad excepcional, pero lo que sí es absolutamente esencial es que sea capaz de reconocer las buenas ideas de diseño de los demás. Antôine de Saint-Exupery afirma que "*La perfección (en el diseño) se alcanza no cuando ya no hay nada que agregar, sino cuando ya no hay nada que sacar."* Así, cuando el código va mejorando y se va simplificando, es cuando se sabe que se está en el camino correcto. A través de la revisión iterativa y constante de la gran cantidad y variedad de desarrolladores es como el código se refina quedando muy simple y eficiente.

5.4.11. Controle su documentación

De acuerdo con Larman (2003) si la comunidad de desarrollo dispone de una herramienta que automatiza el control de las versiones de la documentación de proyectos (a cualquier nivel), esta puede organizarse por *disciplinas* (una carpeta por cada una) tal como se muestra en la Figura 22: Organización de la Documentación por Disciplinas. Al finalizar cada iteración, usualmente se crearía un punto de control etiquetado y congelado de todas las carpetas de la documentación.

Figura 22: Organización de la Documentación por Disciplinas

Figura 23: Organización de la Documentación por Fases

En caso de que la comunidad de desarrollo, no pueda disponer de una herramienta que automatice el control de las versiones, puede organizar la documentación como se sugiere en la Figura 23: Organización de la Documentación por Fases, donde el control de las versiones sobre los *artefactos* (documentos, diagramas, código etc.) del proyecto se realiza de forma manual, creando copias de respaldo al final de cada *iteración* y puntos de control etiquetados

5.5. PRÁCTICAS NO RECOMENDADAS POR AGILE-DISOP

Este numeral tiene como propósito fundamental, prevenir al usuario de **Agile-DISOP** sobre la puesta en práctica de algunos *mitos* y errores comunes relacionados con el desarrollo iterativo e incremental que podrían amenazar el éxito del proyecto y que lamentablemente se han propagado dentro de la comunidad de desarrolladores de software. Estas prácticas “no recomendadas” se enuncian a continuación:

- **Fijar la duración de las iteraciones con fechas de finalización difusas o distantes.** Parkinson (1958) observo con decepción que: *"el trabajo se expande de manera que rellena el tiempo disponible para su finalización".* Cerca del comienzo del proyecto parece ser que existe suficiente tiempo para proceder sin prisa, sin embargo, este efecto se agrava aún más si algunos compromisos tienen fechas de finalización difusas o distantes. Este tipo de planificación, no motivará al equipo de desarrollo a tomar partido y moverse rápido, en cambio, con iteraciones más cortas y definidas, los equipos de desarrollo tendrán una permanente sensación de competencia y entrega al final de cada iteración, creando un clima psicológico de eficacia y confianza en el equipo y por supuesto esta confianza es propagada a los clientes. Este aspecto se relaciona mucho con las estructuras de equipo que participan en proyectos de ejecución táctica, véase el numeral 5.8.1. para más detalles.

- **Todas las iteraciones se planifican en detalle.** Es una tendencia muy generalizada (legado del proceso en cascada), pretender una planificación exhaustiva y especulativa de cada iteración desde el inicio hasta el final del proyecto, indicando para cada iteración: recursos, tiempo, esfuerzo y presupuesto. Lo cierto es que las estimaciones realizadas durante el inicio del proyecto son poco fiables y sólo constituyen una guía sobre si vale la pena realizar una estimación más profunda luego. Es recomendable realizar un poco de trabajo realista antes de generar estimaciones. Aparentemente esta en contradicción respecto al anterior párrafo, sin embargo este y el anterior constituyen los dos extremos del espectro de la planificación.

- **Adopción instantánea del proceso.** Una tendencia muy generalizada y que a menudo genera malestar entre los desarrolladores consiste en querer implantar el proceso completo (con todo el conocimiento y habilidad que ello implica) de un día para otro, desconociendo el nivel de experiencia de los equipos que conforman la comunidad de desarrollo, esta práctica impacta de

forma negativa en la confianza y entusiasmo del equipo, en el sentido en que se asume el proceso como algo complicado, impráctico y en ocasiones hasta inútil. Es importante tener en cuenta que un equipo para el que son nuevas muchas de las prácticas y tecnologías, naturalmente irá más despacio y necesitará más tiempo para comprender el proceso. Por ejemplo: el equipo asumirá un conjunto de prácticas del proceso y en la medida que las domine, podrá ir incorporando más.

Con el propósito de facilitar la comprensión y lectura del presente numeral y según lo recomendado por (McConell 1997) se ha dispuesto en el Anexo 6 un inventario de malas prácticas y errores clásicos adicionales que se relacionan con la planificación y desarrollo de proyectos informáticos. En el mismo sentido, y para conformar **Agile-DISOP**, se contemplaron y contextualizaron (en el marco de una comunidad de desarrollo) malas prácticas en áreas críticas tales como: Personas, Proceso, Producto y Tecnología. Finalmente, debe asumirse que todo lo que sea contrario a las “buenas prácticas” (ver numeral 5.4) es considerado como “mala práctica” aunque no esté contemplada en la tabulación del Anexo 6.

5.6. VISIÓN GENERAL DE LA METODOLOGÍA PROPUESTA

En la Figura 24: Visión General de Agile-DISOP presentada más abajo en este numeral se proporciona un panorama global de la metodología de desarrollo de software **Agile-DISOP** propuesta. Como puede observarse, en la figura, la metodología **Agile-DISOP**, propone dos niveles de ejecución para cualquier emprendimiento de desarrollo de software. En primer lugar, se contempla el nivel “Macro”, en el cuál son determinados los objetivos y alcances del producto a desarrollar en comunidad, todos ellos de alto nivel. En segundo lugar, se contemplan los sub-proyectos de desarrollo, los cuáles pueden ser ejecutados por equipos de (i+d) geográficamente distantes que se integran a la comunidad de desarrollo mediante la plataforma de integración soportada por las TIC y que se recomienda en la práctica descrita en el numeral 5.4.9. Cada sub-proyecto es

ejecutado bajo un ambiente distribuido y tiene sus propias asignaciones respecto al tiempo, presupuesto y recursos que consume, así como las actividades y productos de los cuáles es responsable. Estas asignaciones localizadas en cada sub-proyecto, alimentarían las asignaciones globales de los mismos aspectos para valorar el avance del proyecto en comunidad en el nivel "Macro".

Como puede observarse en la Figura 24, la metodología **Agile-DISOP**, propone unos elementos integradores denominados *"Disciplinas",* las cuáles agrupan basada en su naturaleza, las actividades de la ingeniería del software y gestión de proyectos fomentando el espíritu comunitario a través de todos los equipos de desarrollo y sub-proyectos geográficamente dispersos que ejecuta la comunidad. Las *disciplinas* pueden comportarse como integradoras transversales (a todos los *equipos* de desarrollo en la comunidad), verticales (a todos los sub-proyectos de un macro-proyecto que ejecuta la comunidad) y mixtos (para todos los equipos y proyectos).

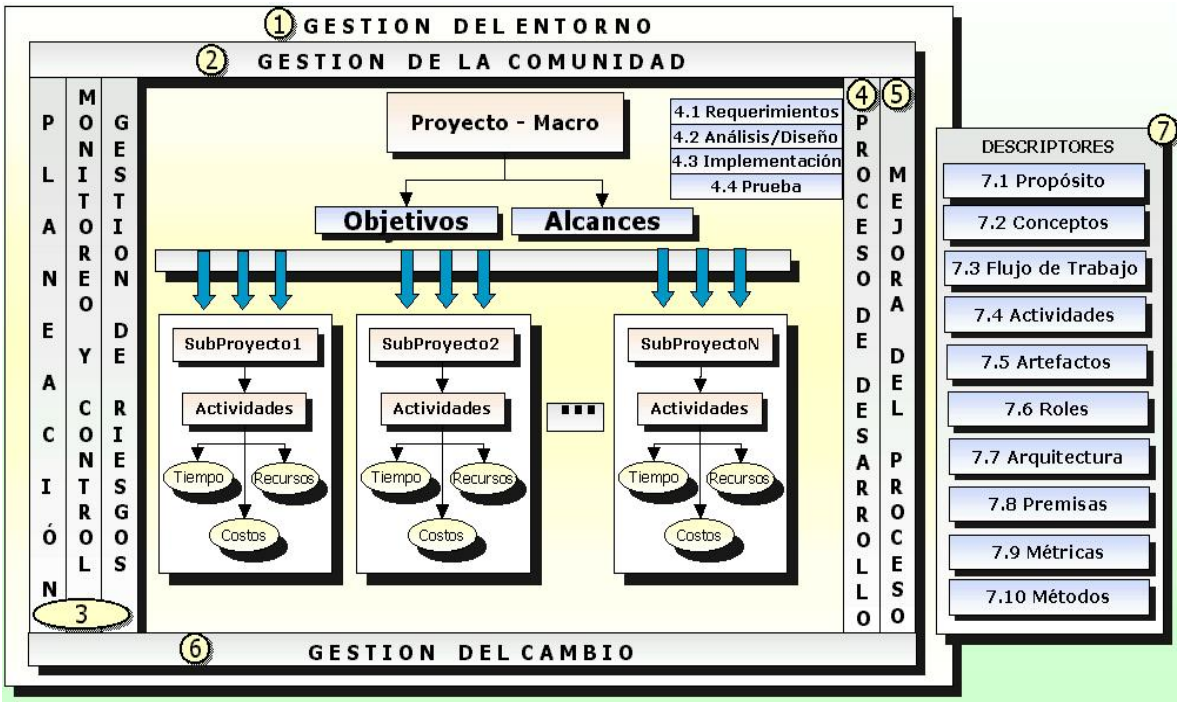

Figura 24: Visión General de Agile-DISOP

Mientras que las *disciplinas* transversales, favorecen la percepción de unidad en la comunidad, las verticales fomentan la percepción de totalidad sobre los sub-proyectos distribuidos en un solo Macro-Proyecto. Así mismo, cada *disciplina* estará soportada por servicios informáticos disponibles en la plataforma de integración seleccionada (véase el numeral 5.15) para dar soporte a la comunidad. Finalmente, la especificación de cada *disciplina* en la Figura 24: Visión General de Agile-DISOP de esta metodología se realizará mediante los descriptores siguientes:

- **Propósito.** Especifica el objeto fundamental de la disciplina.

- **Conceptos.** Consiste en la definición de términos específicos contemplados por **Agile-DISOP**, la cuál es imprescindible para la correcta comprensión de los procedimientos propuestos y la adopción incremental del **Agile-DISOP**. Con el objeto de facilitar la lectura del documento concentrándose en los elementos esenciales, todos los términos implicados en la metodología **Agile-DISOP,** han sido colocados en el Anexo 3.

- **Flujo de Trabajo.** Según RUP (2003), la sola enumeración de todos los *roles, actividades y artefactos* no constituye un proceso. Se requiere una forma de describir una secuencia de actividades que sea significativa, produciendo un resultado tangible y mostrando las interacciones entre los *roles*. En términos de UML, un flujo de trabajo se puede expresar en términos de un *diagrama de secuencia, de colaboración o de actividad*. En este documento se utilizará la forma de un *diagrama de actividad* para describir el flujo de trabajo de cada disciplina.

- **Actividades.** Según RUP (2003), una *actividad* es algo que un rol ejecuta para proveer un resultado significativo en el contexto del proyecto. La *actividad* es considerada como una unidad de trabajo que se utiliza como punto de partida para planear y determinar el progreso del proyecto. Una *actividad* tiene un propósito bien definido, el cuál, se asocia frecuentemente

a la creación o actualización de algunos artefactos. La *granularidad* de una actividad corresponde a unas pocas horas o días y usualmente involucra a un único rol. En consecuencia, este descriptor tiene como propósito, especificar el objeto, entradas, salidas y responsable de cada actividad presente en el flujo de trabajo, mediante una tabla.

- **Artefactos.** Este descriptor tiene como propósito, especificar los *artefactos* que se utilizan y generan dentro de las actividades de la disciplina. Según RUP (2003), un *artefacto* consiste en un producto del proceso. Los roles utilizan a los *artefactos* para ejecutar las actividades y a su vez producen nuevos *artefactos* en el curso de su ejecución. En consecuencia, las actividades tienen artefactos de entrada y de salida. Cada *artefacto* es responsabilidad de un único *rol*, de esta forma es fácil identificar, entender y promover la idea de que cada pieza de información producida en el proceso, requiere un conjunto específico de habilidades. En ocasiones aunque un *rol* "posee" la responsabilidad sobre un *artefacto*, otros *roles* pueden usarlo y hasta actualizarlo si tienen la autorización para hacerlo. Los *artefactos* pueden ser:

    1. **Un modelo.** Por ejemplo, un modelo de *casos de uso* o un modelo de *clases del diseño*, el cuál a su vez puede contener a otros *artefactos*.

    2. **Un elemento contenido en un modelo.** Por ejemplo, un *caso de uso o una clase del diseño*.

    3. **Un elemento de la implementación**. Por ejemplo, una Base de datos, el código fuente o ejecutable.

    4. **Los documentos.** Por ejemplo, una especificación de *caso de uso*, o un plan de proyecto. Se proveerá el estándar en un anexo cuando el artefacto corresponda a un documento.

- **Arquitectura.** Consiste en un *diagrama de clases* del análisis (sin atributos de clase) en notación de UML que describe la estructura de la *disciplina* en términos de sus elementos e interrelaciones.

- **Premisas.** Describe las condiciones y normas de comportamiento que son aceptables en la comunidad de desarrollo durante la ejecución de las actividades de la *disciplina*. No es aplicable a todas las *disciplinas,* en este caso no aparecerá el numeral correspondiente en la descripción de la disciplina.

- **Métricas.** Sugiere algunas *métricas* aplicables dentro de la *disciplina*, bien sea para elaborar los artefactos o valorar su calidad. No es aplicable a todas las *disciplinas,* en este caso no aparecerá el numeral correspondiente en la descripción de la disciplina.

- **Métodos**. Enuncia los métodos específicos para la elaboración de los *artefactos* de la *disciplina*. No es aplicable a todas las *disciplinas,* en este caso no aparecerá el numeral correspondiente en la descripción de la disciplina.

5.7. ORGANIZACION DEL PROCESO

En la

Figura 25: Visión Detallada del Agile-DISOP, se puede observar más concretamente los elementos internos del **Agile-DISOP** desde dos perspectivas. En primer lugar, el eje horizontal representa el tiempo e ilustra el aspecto dinámico de la metodología expresado en términos de *fases, iteraciones e hitos*. En segundo lugar, el eje vertical, representa el aspecto estático expresado en términos de las *disciplinas* (compuestas por sus *flujos de trabajo, artefactos y roles*) y su correspondiente nivel

de énfasis (medido en unidades de esfuerzo[61]) el cuál está representado por las curvas de área que varían según su intersección con cada fase. Las *disciplinas* de mejoramiento del proceso y gestión de la comunicación tienen un énfasis que no es representable por las curvas de área, puesto que se encuentran en un nivel superior al del producto de software. Se hace énfasis en que **Agile-DISOP**, se inspira en el proceso unificado de desarrollo, enriquecido con algunas prácticas y recomendaciones de las metodologías de desarrollo de software libre.

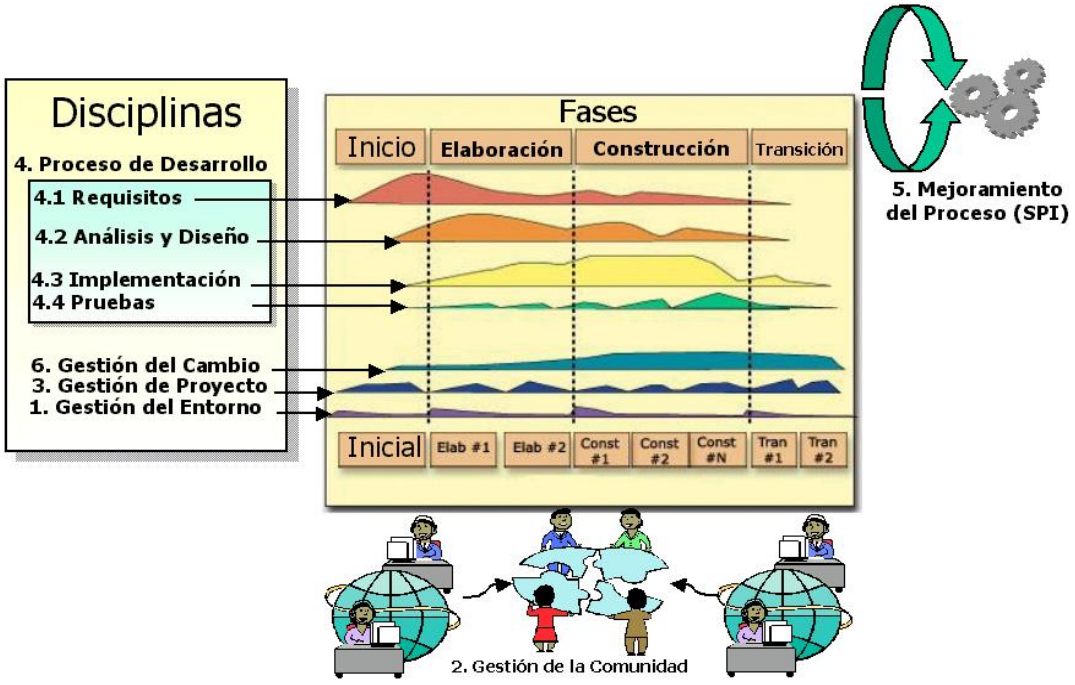

Figura 25: Visión Detallada del Agile-DISOP

A continuación se detalla brevemente la organización del proceso en términos de las *disciplinas, fases y roles* (organizados en *equipos*).

5.7.1. Disciplinas del Agile-DISOP

[61] Por ejemplo: Horas Hombre al Mes.

En este numeral, se describen brevemente las *disciplinas* contempladas por **Agile-DISOP** en la misma secuencia que se muestra en la Figura 24 y

Figura **25**:

1. **Gestión del Entorno.** Su propósito consiste en fortalecer la adecuación, puesta en marcha y mantenimiento tanto del proceso mismo **Agile-DISOP** como de la plataforma de integración para la comunidad de desarrollo seleccionada. El numeral 5.9 trata más a fondo este elemento de **Agile-DISOP.**

2. **Gestión de la Comunidad**. Su propósito consiste en soportar las necesidades de comunicación, coordinación y gestión de la disponibilidad para personas y equipos (i+d) bajo un ambiente de dispersión geográfica que pertenecen a la comunidad de desarrollo. El numeral 5.10 trata más a fondo este elemento de **Agile-DISOP.**

3. **Gestión del Proyecto.** Su propósito consiste en proveer un marco metodológico para la planificación, monitoreo, control y gestión de riesgos en macro-proyectos y/o sub-proyectos informáticos de la comunidad de desarrollo. El numeral 5.11 trata más a fondo este elemento de **Agile-DISOP.**

4. **Proceso de Desarrollo.** Su propósito consiste en proveer un marco metodológico para la ejecución de las *disciplinas* orientadas hacia la investigación de requisitos-análisis, diseño, implementación y pruebas para el desarrollo software. El numeral 5.12 trata más a fondo este elemento de **Agile-DISOP.**

5. **Mejora del Proceso (SPI).** Su propósito consiste en proveer los mecanismos para soportar el mejoramiento continuo de mismo modelo **Agile-DISOP** con el propósito de alcanzar de forma incremental un grado de

madurez cada vez más alto y competitivo. El numeral 5.13 trata más a fondo este elemento de **Agile-DISOP.**

6. **Gestión del Cambio.** Su propósito consiste en proveer mecanismos que soportados informaticamente permitan controlar los cambios en los artefactos producidos por la comunidad de desarrollo. El control sobre la documentación, ayudará a la comunidad a evitar la confusión, asegurando que los artefactos resultantes no tengan conflictos entre sí, debido a las actualizaciones simultáneas y las numerosas versiones que puede manejar la comunidad de desarrollo en un momento determinado. El numeral 5.13.1 trata más a fondo este elemento de **Agile-DISOP.**

5.7.2. Fases del Agile-DISOP

Las fases en **Agile-DISOP** se entienden como los momentos que vive el software a través de su maduración desde una idea hasta un producto terminado durante el ciclo de vida de desarrollo. Según RUP (2003) se proponen 4 de estas fases: Inicio, Elaboración, Construcción y Transición. Cada fase una tiene un propósito y unos hitos que determinan cuando la fase termina, dando paso a la siguiente. **Agile-Disop**, retoma el espíritu de estas fases de RUP las cuáles se describen brevemente a continuación:

5.7.2.1. Fase de Inicio

Su objetivo fundamental consiste en alcanzar la estabilidad de los objetivos del sistema mediante el consenso general de toda la comunidad de desarrollo. Durante esta fase, la comunidad de desarrollo debe ponerse de acuerdo en cuestiones relativas a los alcances, riesgos, requerimientos y objetivos del sistema con el propósito de comprobar la viabilidad del proyecto teniendo en cuenta las condiciones del entorno de desarrollo. Durante esta fase es preciso determinar lo siguiente:

- **El alcance del proyecto.** Determina las características que estarán en el producto y las que no, mediante la captura de los requerimientos más importantes y las restricciones del sistema.

- **Proponer una arquitectura candidata.** Los casos que son críticos desde el punto de vista de la arquitectura, ayudarán a seleccionar un estilo arquitectónico. De otro lado, la valoración de que componentes se van a desarrollar, reutilizar o comprar ayudará a la estimación de los costos y recursos para el proyecto. Se sugiere el desarrollo de un prototipo rápido que ayude a la estimación de la viabilidad del proyecto mismo.

- **Estimar en términos generales el costo, cronograma y riesgos.** Para el proyecto entero haciendo más énfasis en la fase de "Elaboración" que sigue.

- **Preparar el entorno para soportar la ejecución del proyecto en comunidad.** Valorando y ajustando el proyecto Vs. la comunidad, se seleccionan las herramientas y se decide que partes del proceso se van a adoptar.

Hito de la Fase de Inicio. La estabilización de los objetivos y alcances del proyecto determinan la finalización de la fase de inicio, en este punto la viabilidad del proyecto es clara y puede decidirse si es favorable continuar aceptando el riesgo o por el contrario lo mejor es cancelar el proyecto entero. Los criterios para determinar si se ha llegado hasta este punto son:

- **Consenso general del equipo promotor y la comunidad de desarrollo**. Sobre el alcance, costo y cronograma para el proyecto así como el esclarecimiento y entendimiento compartidos de un conjunto correcto de requerimientos para el sistema. Finalmente un acuerdo global sobre los ajustes al proceso de desarrollo que se va a utilizar durante la ejecución del proyecto en comunidad así como de las herramientas de soporte.

- **Identificación de todos los riesgos.** Y sus correspondientes estrategias de mitigación.

5.7.2.2. Fase de Elaboración.

Esta fase del proceso, tiene como propósito fundamental la implementación de los requisitos funcionales que son críticos desde el punto de vista de la arquitectura, haciendo realidad la estabilización temprana de misma. De otro lado, esta fase, determinará una concepción estable de los requerimientos y las estrategias relacionadas con la mitigación de riesgos para estimar de manera más fiable los costos y cronogramas para la culminación del proyecto. Durante esta fase, la comunidad debe ponerse de acuerdo sobre varias cuestiones riesgosas tales como: conflictos de requerimientos y diseño, reutilización de componentes, estrategias de demostración del producto a los clientes, asegurar una línea base de la arquitectura que soportará los requerimientos del sistema en un tiempo y costo razonables, establecer y configurar un entorno de soporte, interacción e integración para la comunidad. Durante esta fase es preciso realizar lo siguiente:

- **Definir, validar y estabilizar una arquitectura de forma rápida y estable.** Mediante el refinamiento continuo del conocimiento relacionado con el problema, se va conformando un conocimiento sólido relacionado con los casos de uso más críticos que dirigen las decisiones arquitectónicas y de planeación.

- **Crear una base de planes detallados para las iteraciones durante la fase de construcción.**

- **Configuración y puesta en marcha del entorno de desarrollo.** Incluye el proceso, las herramientas y la plataforma de integración requeridos para soportar la comunidad de desarrollo.

- **Refinamiento de la Arquitectura y selección de componentes.** Los componentes potenciales que integrarían la arquitectura son evaluados e involucrados en decisiones de reutilización, desarrollo o compra con el propósito de estimar con más confianza los costos y cronogramas de la siguiente fase de construcción. Los componentes seleccionados son integrados y evaluados contra los escenarios más importantes. La interpretación de los resultados de evaluación obtenidos pueden motivar un rediseño de la arquitectura.

Hito de la Fase de Elaboración. La estabilización del ciclo de vida de la arquitectura establece una base para que la comunidad de desarrollo pueda escalar sobre ella durante la siguiente fase de construcción al implementar los casos de uso que no son críticos arquitectónicamente hablando. En este punto, los objetivos y el alcance del sistema han sido examinados, un estilo arquitectónico se ha seleccionado y los riesgos de mayor impacto han sido resueltos. Los criterios para determinar si se ha llegado hasta este punto son:

- **La visión del producto, sus requerimientos y su arquitectura se han estabilizado.** Además toda la comunidad infiere que la visión se alcanzará si el plan de proyecto se ejecuta implementando el producto en el contexto de la arquitectura base.

- **Las estrategias de prueba y evaluación se han aprobado.**

- **Los planes de iteración para la fase de construcción están lo suficientemente detallados** y estimados para permitir que el proyecto pueda proceder de forma confiable.

- **La diferencia entre el consumo real de los recursos es aceptable respecto del estimado.**

5.7.2.3. Fase de Construcción.

Esta fase del proceso, tiene como propósito fundamental el análisis, diseño, implementación y prueba del resto de casos de uso (y otros requerimientos) que agregan funcionalidad al sistema, los cuáles crecen sobre la línea base de la arquitectura estabilizada durante la anterior fase de elaboración. En cierto sentido, esta fase se parece a un proceso de manufactura, debido a que se concentra en la gestión de recursos y operaciones para optimizar los costos, cronogramas y la calidad durante la ejecución del resto del proceso. Además esta fase se concentra en la determinación de la disposición de la comunidad de usuarios para recibir las primeras versiones de producción del software. De otro lado, es preciso determinar los mecanismos que permitan el trabajo paralelo para los equipos de desarrollo de la comunidad, siempre que los recursos lo permitan, con el propósito de incrementar la velocidad de desarrollo pero teniendo en cuenta el aumento en la complejidad de la gestión. Sin embargo, una arquitectura robusta influye en la asignación de trabajo paralelo. Durante esta fase es preciso realizar lo siguiente:

- Realizar una gestión, control y optimización de los recursos y procesos.
- Completar el desarrollo y prueba de los componentes, requisitos funcionales y no funcionales restantes.
- Evaluación de las versiones del producto que se han liberado respecto de su visión.

Hito de la Fase de Construcción. La capacidad operacional inicial, consiste en la certeza de que el producto está listo para ser liberado pues ya se ha implementado toda la funcionalidad y al menos en parte esta ha sido probada. Por otra parte, al menos existe un manual de usuario sobre la versión actual que esta disponible. Los criterios que indican si se ha alcanzado el final de esta fase son:

- El producto es lo suficientemente maduro y estable como para distribuirlo a la comunidad de usuarios?

- Todos los interesados están preparados para recibir la próxima fase de transición.

- La diferencia entre el consumo real de los recursos es aceptable respecto del estimado.

5.7.2.4. Fase de Transición.

El propósito fundamental de esta fase consiste en asegurar que el software estará disponible para sus usuarios finales. Para iniciar la fase de transición, es necesario que la línea base de funcionalidad del producto este madura y sea suficiente para ser distribuida a través de la comunidad de desarrollo. Esto último implica que usualmente exista un subconjunto de características del sistema implementadas y documentadas con calidad, listas para ser aprovechadas por los usuarios finales.

Durante la transición es posible iterar varias veces haciendo énfasis en la disciplina de pruebas con el objeto de preparar el producto para su liberación con base en la realimentación de los usuarios. En este punto, la realimentación del usuario ayuda a enfatizar en los ajustes de granularidad fina sobre el producto, su configuración e instalación, así como las dificultades relacionadas con la usabilidad. Al final de la fase de transición el proyecto debe estar próximo a finalizar, sin embargo el final del proyecto puede coincidir con el inicio de otro para la próxima versión del producto. También es posible que todos los artefactos del proyecto sean entregados a otro equipo encargado del mantenimiento del software durante su etapa de producción. La naturaleza del producto desarrollado determina la complejidad de esta fase. Las actividades de esta fase suelen ajustarse al objetivo perseguido para la

transferencia del producto, ejemplo: si se está depurando los errores, tan solo es necesario ejecutar las disciplinas de implementación y prueba[62], sin embargo si se requiere agregar nuevas características, entonces debe ejecutarse también las disciplinas del análisis y el diseño.

De otra parte, es importante que se socialice a toda la comunidad de desarrollo las características importantes ofrecidas por la nueva versión del producto así como un informe ejecutivo de los errores corregidos. Finalmente es importante delimitar las estrategias relacionadas con el entrenamiento de los usuarios finales y puesta en marcha del plan de marketing y distribución así como el ajuste a los errores en el desempeño, usabilidad del producto y soporte al usuario. Durante esta fase es preciso realizar lo siguiente:

- Ejecutar el plan de despliegue del producto.
- Finalizar el material de soporte para el usuario final.
- Realizar pruebas al producto que se distribuirá en el entorno de desarrollo.
- Crear una versión del producto para liberar y obtener re-alimentación de los usuarios. Posteriormente el producto se ajustará con base en esa re-alimentación.
- Finalmente, hacer que el producto esté disponible para los usuarios de la comunidad.

Hito de la Fase de Transición. Al final de esta fase debe valorarse el estado de satisfacción de los objetivos del sistema y si debe iniciarse un nuevo ciclo de desarrollo para una nueva versión de producción del software, es posible que este

---

[62] Ejemplo: Pruebas beta del nuevo sistema contra las expectativas de los usuarios y operación en paralelo con el sistema que será reemplazado.

hito coincida con el final de la fase de inicio para el próximo ciclo de desarrollo. Finalmente este hito es alcanzado cuando existe una revisión y aceptación de la distribución del producto por parte de los usuarios finales y los recursos-tiempo consumidos son aceptables respecto a lo planificado.

En la Figura 26, se ilustra la distribución sugerida (RUP 2003) del esfuerzo y el tiempo dedicados durante cada una de las fases anteriormente descritas.

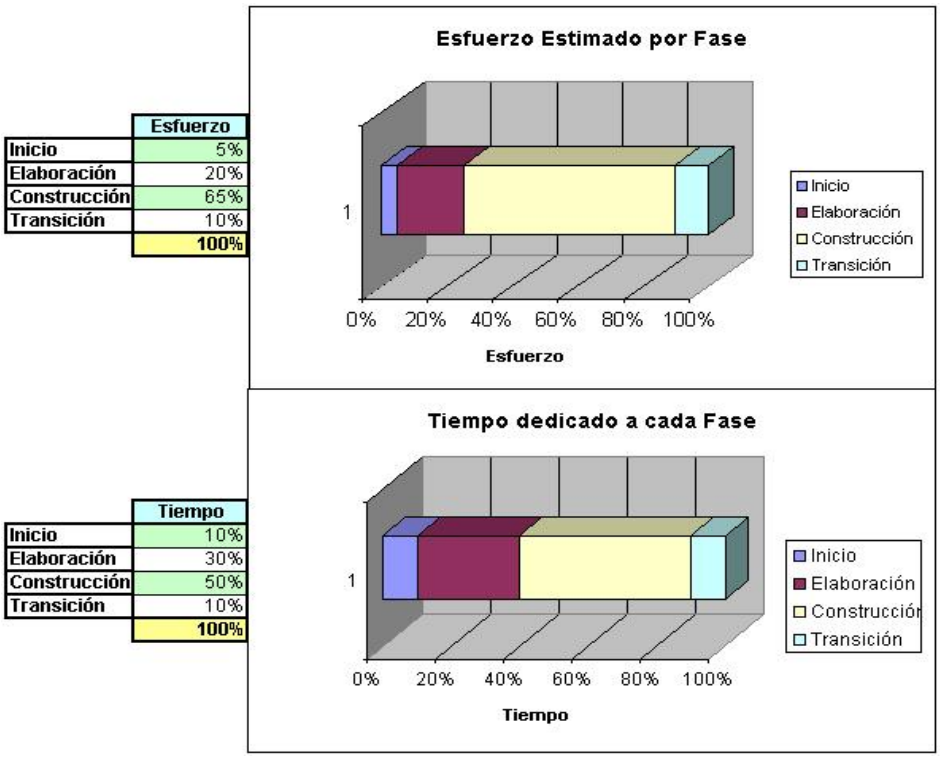

Figura 26: Estimación del Esfuerzo y Duración de las Fases

Una comunidad de desarrollo está compuesta en su esencia más profunda e importante por las personas. En consecuencia, es preciso determinar la topología de organización para la comunidad de desarrollo. En el siguiente apartado se proporciona una panorámica que intenta visionar las configuraciones de equipo versus los tipos de proyectos posibles. Finalmente se tiene en cuenta las

interacciones de comunicación involucradas en una comunidad de desarrollo y se sugiere una estructura de equipo para la misma.

5.8. EQUIPO DE DESARROLLO EN COMUNIDAD

Según MacConnell (1997) un *equipo* se define como: *"un conjunto de personas pequeño con habilidades complementarias, que están comprometidas en un propósito, objetivos de rendimiento y enfoque comunes en el que todos y cada uno de los miembros del equipo es responsable ante los demás".* Un *equipo* existe siempre que dos o más cabezas juntas funcionan mejor que por separado. En consecuencia, no todos los grupos de personas son *equipos* porque no tienen sinergia en ellos. La sinergia implica un compromiso profundo e incondicional de cada miembro del *equipo* que fomenta un alto sentido de identidad. Los miembros de un *equipo* verdadero, se caracterizan por que trabajan duro y unidos, se divierten con su trabajo y pasan gran parte del tiempo centrándose en los objetivos del proyecto. Al contrario, muchos "falsos equipos", están integrados por miembros desmoralizados y dispersos, que trabajan la mayor parte del tiempo en objetivos opuestos a los del proyecto. Lakhanpal (1993) descubrió que la unión del grupo, contribuía más a la productividad que las capacidades o experiencias individuales de los miembros del proyecto. Sin embargo los directores de proyecto, escogen a los miembros del *equipo* basándose en el nivel de experiencia individual en lugar de validar su capacidad para conformar un *equipo* verdaderamente unido que crea sinergia con su trabajo. En conclusión un *equipo* productivo y sinergético se forma a partir de una sólida cohesión y colaboración entre sus miembros. De acuerdo con McConell (1993), un *equipo* verdaderamente productivo debe satisfacer los siguientes requisitos:

- **Visión**. Los miembros del *equipo* comparten una alta visión manifestando un fuerte sentido de identidad, al mismo tiempo, disfrutan y se enriquecen con su trabajo.
- **Compromiso.** Los miembros del *equipo* están muy comprometidos y son competentes en sus responsabilidades particulares.
- **Confianza.** El *equipo* refleja una fluida interdependencia que está fundamentada sobre la confianza mutua y comunicación efectiva de sus miembros.
- **Resultados.** El *equipo* se basa sobre una estructura dirigida por resultados, esta integrado por pocos miembros y posee una saludable autonomía para tomar decisiones importantes rápidamente.

En el siguiente numeral, se profundiza en la reflexión sobre la configuración y estructura de los equipos que podrían integrar a la comunidad.

5.8.1. Configuración y Estructura de Equipos

Determinar el objetivo del *equipo* es fundamental para configurar la estructura correcta del mismo. Según Larson y LaFasto (1989) existen tres estructuras básicas de equipo dependiendo del objetivo que persiguen y estas son:

- **Estructura de Equipo Centrado en la Resolución de problemas.** Se enfoca en la resolución de un problema complejo y/o poco definido. Sus miembros, están ocupados en una o más cuestiones específicas. Sus miembros deben ser confiables, inteligentes y pragmáticos. **Ejemplo:** Un grupo de programadores de mantenimiento intentando diagnosticar un nuevo defecto del software.

- **Estructura de Equipo centrado en la Creatividad**. Este *equipo* explora nuevas posibilidades y alternativas para un producto de software. Sus miembros necesitan estar motivados, ser independientes, creativos y

persistentes. La estructura del *equipo* necesita soportar la autonomía individual y colectiva de los miembros del *equipo*. **Ejemplo:** Un grupo de programadores que esta en proceso de concepción de nuevas alternativas de interfaz gráfica de usuario para una aplicaciones multimedia.

- **Equipo centrado en la Ejecución Táctica.** Este *equipo* se centra en ejecutar un plan bien definido el cuál especifica tareas y funciones concretas. Sus miembros necesitan tener un sentido de la urgencia de su misión, estar más interesados en la acción que en la intelectualización del problema. Para evaluar su desempeño, este tipo de *equipo* contara con factores críticos de éxito claros. **Ejemplo:** Un *equipo* de software trabajando sobre una actualización bien definida de un producto, en la que el propósito de la actualización no es definir nuevas alternativas para el producto, sino poner características bien conocidas en manos de los usuarios tan pronto como sea posible.

A continuación, se describen las configuraciones de *equipo* disponibles para las estructuras arriba mencionadas.

5.8.1.1. Equipo de Negocios.

Constituido por un grupo de iguales encabezados por un jefe técnico. Aparte del jefe técnico todos los miembros del *equipo* tienen el mismo estatus, diferenciándose en su ámbito de experiencia: base de datos, gráficos, interfaces de usuario y lenguajes de programación. El *jefe técnico* contribuye de forma activa y es contemplado como el primero de todos, además es elegido según la experiencia técnica y no por su eficacia como gestor. De otra parte, este es responsable de tomar las decisiones finales sobre cuestiones técnicas. En ocasiones, el *jefe técnico* es un miembro normal del *equipo*, que simplemente tiene la tarea extra de ser el enlace entre el *equipo* y la directiva. En otros casos, este ocupa una posición directiva de primer nivel. Respecto a las responsabilidades, el *jefe técnico* permite a cada miembro del

equipo trabajar en su área de experiencia y permite que el propio *equipo* decida en qué va a trabajar cada uno de ellos. Esta configuración, funciona bien con grupos pequeños y que llevan juntos mucho tiempo o que pueden estructurar sus relaciones con el tiempo. Finalmente, es bastante adaptable de trabajar en todos los tipos de proyectos: *resolución de problemas, creatividad, ejecución táctica*. Pero generalmente ésta es su debilidad, y en muchos casos hay otra estructura que puede funcionar mejor. La Figura 27 ilustra la configuración del *equipo* de negocios.

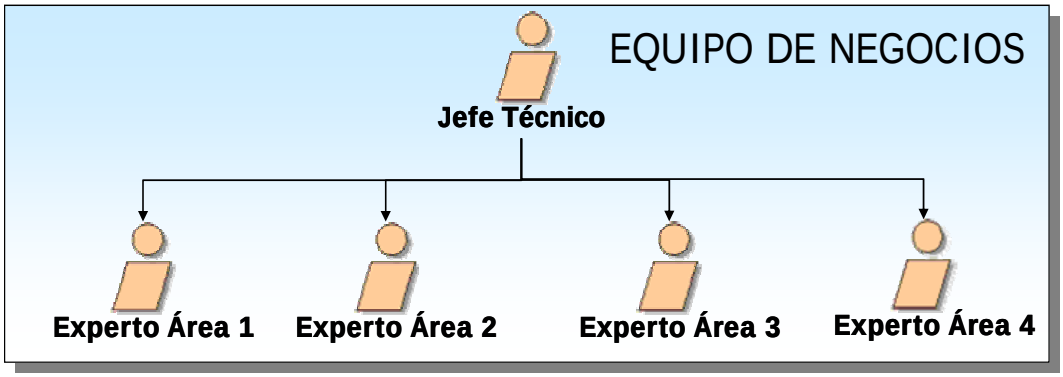

Figura 27: Estructura Organizacional del Equipo de Negocios

5.8.1.2. Equipo con Programador Jefe.

El *equipo* con *programador jefe* saca partido del hecho de que algunos desarrolladores son más productivos que otros. Las estructuras normales de *equipo* ponen al mismo nivel a programadores mediocres y a grandes estrellas. Se saca partido de la productividad de las grandes estrellas, pero se sufre la penalización de la baja productividad del resto de miembros del *equipo*. El *programador jefe* realiza la especificación completa, hace todo el diseño, escribe la mayoría del código de producción y es el responsable final de prácticamente todas las decisiones del proyecto. El resto de los miembros del *equipo* son libres para especializarse con el propósito de apoyar al *programador jefe*. Esta configuración de *equipo*, saca partido del hecho de que los especialista tienden a rendir mejor que los generalistas (Jones, 1994). Sus roles principales son:

- **Programador de Reserva.** Apoya al *programador jefe* como crítico, ayudante de la investigación, contacto técnico con grupos externos y por supuesto programador de reserva.

- **Administrador.** Gestiona cuestiones administrativas, como el dinero, las personas, el espacio y los equipos. Aunque el *programador jefe* tiene la última palabra en estas cuestiones, el administrador le libera de tener que tratarlas de forma diaria.

- **Programador Herrero.** Es el responsable de crear herramientas personalizadas solicitadas por el *programador jefe*. Estaría a cargo de crear secuencias de órdenes y archivos ejecutables, de crear macros para su uso en el editor de programas y de ejecutar la construcción diaria.

- **Abogado del Lenguaje.** Responde a preguntas muy técnicas sobre el leguaje de programación que está usando el *programador jefe*.

El *equipo* con *programador jefe* resulta adecuado para proyectos creativos, en los que tener un cerebro al frente ayudará a proteger la integridad conceptual del sistema. También resulta adecuado para proyectos de ejecución táctica, en los que el *programador jefe* puede ser una especie de dictador que diseñe los medios más expeditivos para lograr la culminación del proyecto. La Figura 28, ilustra la estructura organizacional de esta configuración de equipo.

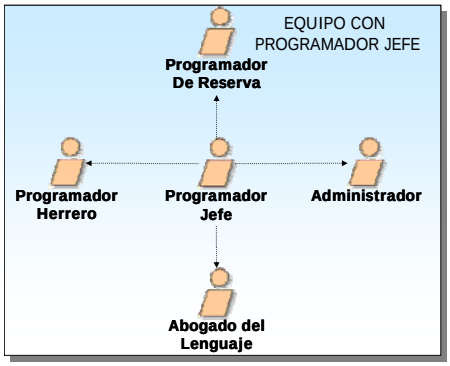

Figura 28: Estructura Organizacional del Equipo con Programador Jefe

5.8.1.3. Equipo en la sombra.

Agrupa a un grupo talentoso de desarrolladores de productos creativos y les pone en una situación en la que son liberados de las restricciones burocráticas habituales de la organización, otorgándoles libertad para desarrollar e innovar. Estos *equipos* son tratados como cajas negras por sus directivos. La directiva no quiere conocer los detalles sobre cómo realizan el trabajo, sino que sólo quieren saber que lo están haciendo. Así, el *equipo* es libre de organizarse como mejor le parezca. Con el paso de tiempo, puede aparecer un líder natural, o el equipo podría designar a un líder desde el principio. Los proyectos en la sombra tienen la ventaja de crear una sensación de intensa propiedad y compromiso por parte de los desarrolladores implicados. El efecto motivacional puede ser impresionante. Tienen la desventaja de no ofrecer mucha visibilidad del progreso del equipo, característica inherente en los trabajos altamente creativos. De otra parte se sacrifica la visibilidad por rescatar la motivación. Los *equipos* en la sombra resultan más adecuados para proyectos exploratorios en los que la creatividad es absolutamente importante. Los *equipos* en la sombra son raramente la estructura más rápida cuando se necesita resolver un problema claramente definido o cuando se necesita ejecutar un plan bien entendido. La Figura 29, ilustra la estructura organizacional de esta configuración de equipo.

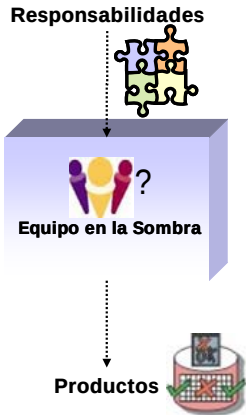

Figura 29: Estructura Organizacional del Equipo en la Sombra

5.8.1.4. Equipo de Prestaciones.

En un equipo de prestaciones, el desarrollo, el control de calidad, la documentación, la gestión del programa y el marketing están organizados con las estructuras jerárquicas tradicionales de responsabilidad. La gente de marketing informa a los responsables de marketing, los desarrolladores a los responsables de desarrollo, etc. Con esta organización tradicional se encuentra los equipos que toman uno o más miembros de cada uno de estos grupos y les asignan la responsabilidad de una parte de la funcionalidad del producto (MacCarthy, 1995). Podría tener un equipo de prestaciones dedicado a la impresión, generación de informes o la generación de gráficos. El equipo de prestaciones se hace responsable de las decisiones tomadas sobre esa parte del producto. Los equipos de prestaciones presentan las ventajas de potenciación, facilidad de seguimiento y equilibrio, que se explican a continuación

- **Potenciación**. El equipo puede potenciar sensiblemente porque incluye representantes de desarrollo, control de calidad, documentación, gestión del programa y marketing; es decir, de todas las partes implicadas.

- **Facilidad de Seguimiento.** El equipo considerará todos los puntos de vista necesarios en sus decisiones, y por ello rara vez dará pie a que estas sean cuestionadas. Tienen acceso a todas las personas que necesitan para tomar buenas decisiones.

- **El equipo está equilibrado.** Pueden obtenerse decisiones equilibradas de un grupo que incluye representantes de las secciones de desarrollo, marketing o control de calidad evitando que cualquiera de estos grupos tenga exclusivamente la última palabra sobre la especificación de un producto.

Los equipos de prestaciones resultan adecuados para proyectos de resolución de problemas, ya que ofrecen el refuerzo y la facilidad de seguimiento necesarios para

resolver las cuestiones planteadas de forma expeditiva. También son adecuados para proyectos de creatividad, ya que la composición interdisciplinaria del equipo puede estimular las ideas. La gestión adicional generada por los equipos de prestaciones se desperdicia en los proyectos de ejecución táctica: si todas las tareas están claramente identificadas, los equipos de prestaciones tienen poco que aportar. En la Figura 30, se ilustra la estructura organizacional de esta configuración de equipo.

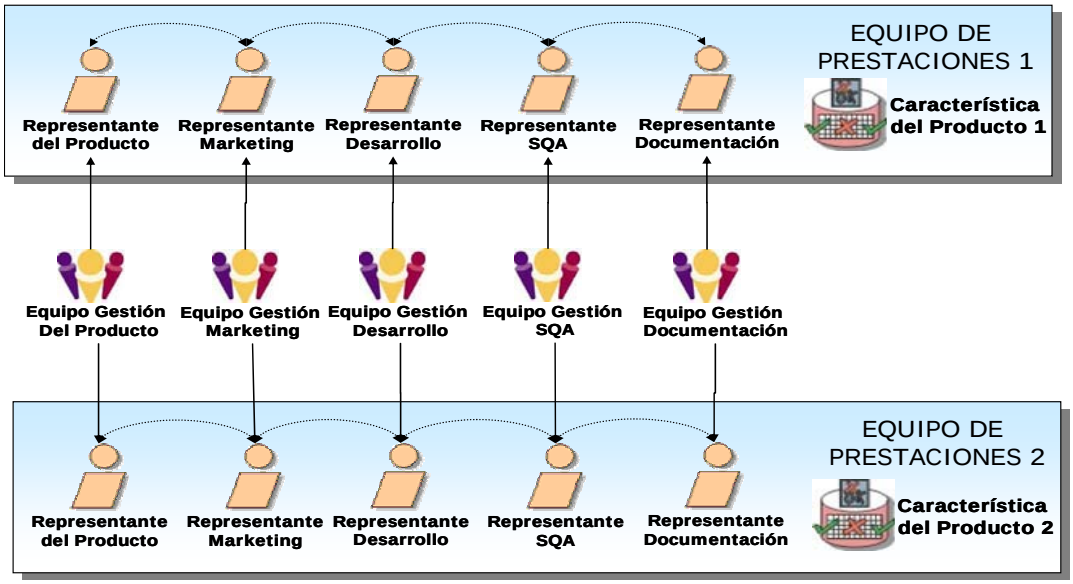

Figura 30: Estructura Organizacional del Equipo de Prestaciones

5.8.1.5. Equipo de Búsqueda y Rescate.

Combina un conocimiento especializado de herramientas software, hardware y un entorno comercial determinado. El *equipo* de búsqueda y rescate es el más apropiado para *equipos* que necesiten centrarse en la resolución de un problema. Está demasiado dirigido a problemas básicos como para soportar mucha creatividad, y está demasiado orientado a corto plazo como para soportar la ejecución táctica. La Figura 31, ilustra la estructura organizacional de esta configuración de *equipo*.

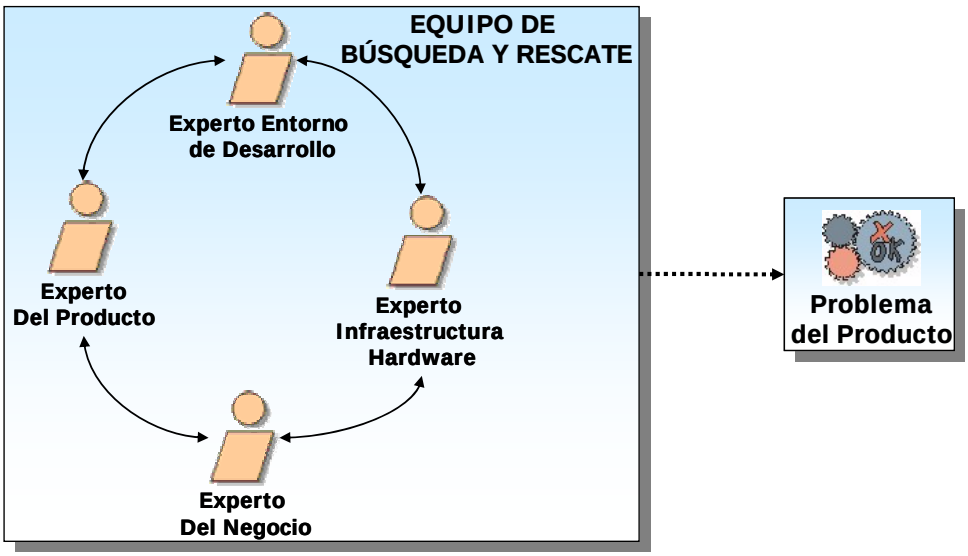

Figura 31: Estructura Organizacional del Equipo de Búsqueda y Rescate

5.8.1.6. Equipo SWAT.

En software SWAT significa “experiencia con herramientas avanzadas” (Skilled with advanced tools). El término nació como parte de la metodología RAD (Martin, 1991). La idea básica consiste en escoger un grupo de personas con una sólida formación en una herramienta o método determinado, y dedicarlas a un problema que se adapta perfectamente a ser resuelto con esa herramienta o método. Un equipo SWAT puede especializarse en una de las siguientes áreas: un paquete determinado de base de datos, un entorno de programación determinado o un método de desarrollo. Los *equipos* SWAT son permanentes, no todo el tiempo trabajan como SWAT, sin embargo están habituados a trabajar juntos y tienen papeles bien definidos. Resultan especialmente adecuados en proyectos de ejecución táctica. Su trabajo no consiste en ser creativos, sino en implementar una solución dentro de los límites de una herramienta o metodología que conocen perfectamente. Funcionan también en proyectos de resolución de problemas. Los miembros del *equipo* tienen confianza mutua y su concentración en una fase determinada el proyecto les permite tratar la realización de esa fase como un único

problema que pueden superar rápidamente. La Figura 32, ilustra la estructura organizacional de un equipo SWAT.

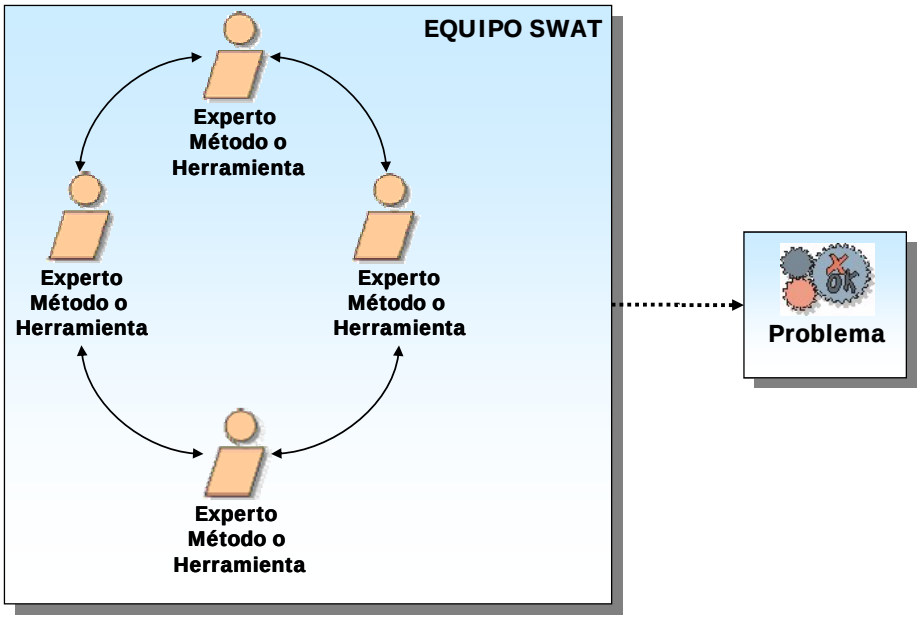

Figura 32: Estructura Organizacional del Equipo SWAT

5.8.1.7. Equipo de Especialistas.

En esta configuración de *equipo* existe un responsable del desarrollo de software que no necesariamente requiere habilidades en programación y debe asegurar un camino despejado para que los desarrolladores trabajen de forma eficiente. Los programadores podrían desarrollar un producto sin el responsable, pero el responsable no podría desarrollar un producto sin los programadores. En esta configuración de *equipo* existen roles altamente especializados y personal cuidadosamente seleccionado. Las especialidades del software pueden incluir: arquitecturas de sistemas, reutilización, evaluación de paquetes, hardware, entornos software, lenguajes de programación, redes LAN, herramientas CASE, soporte al cliente, pruebas y muchas más. Este modelo específico se aplica mejor a los proyectos de ejecución táctica, que hacen énfasis en los papeles altamente

especializados que desarrollan los desarrolladores individuales. La Figura 33, ilustra la estructura organizacional de esta configuración de *equipo*.

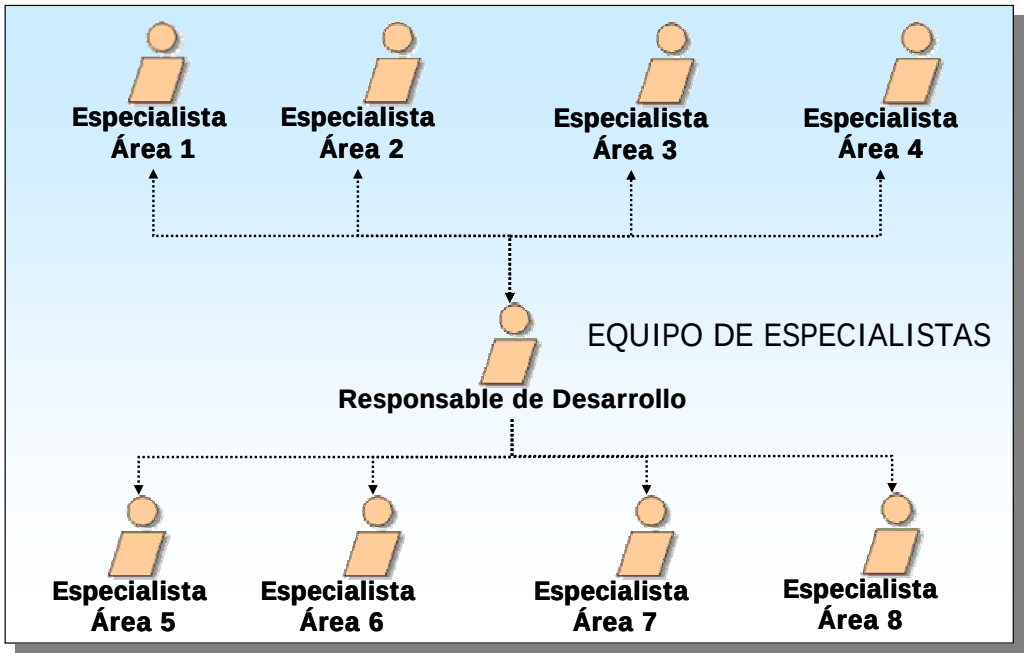

Figura 33: Estructura Organizacional del Equipo de Especialistas

5.8.1.8. Equipo de Teatro.

Esta caracterizado por una fuerte dirección y una gran negociación de los papeles del proyecto. El papel central del proyecto es ocupado por el *director*, que mantiene la visión del producto y asigna a la gente responsabilidades en áreas individuales. Los participantes individuales pueden adaptar sus papeles, sus partes del proyecto, como les dicten sus propios instintos artísticos. Pero no pueden llevar demasiado lejos sus propias ideas, evitando colisionar con la visión del *director*, la visión del director tiene que prevalecer para la buena marcha del proyecto. Cada participante participa en una audición, y negocia un papel. El responsable del software o productor debe obtener la financiación, coordinar las planificaciones, y asegurase de que todo el mundo está en el sitio correcto en el momento adecuado. Generalmente

el productor no juega un papel activo en los aspectos artísticos del proyecto. La fuerza del modelo de teatro consiste en que proporciona una vía para integrar importantes contribuciones individuales junto con una fuerte visión central en proyectos de creatividad. El modelo de teatro resulta particularmente apropiado para equipos software dominados por fuertes personalidades. El modelo de teatro es un modelo adecuado para proyectos multimedia modernos. Mientras anteriormente los proyectos software necesitaban integrar las contribuciones de varios desarrolladores de software, ahora tiene que integrar las contribuciones de diseñadores gráficos, escritores, productores de video, productores de audio, editores, ilustraciones, coordinadores de contenido y varios desarrolladores de software. En la Figura 34 se ilustra la estructura organizacional de esta configuración de equipo.

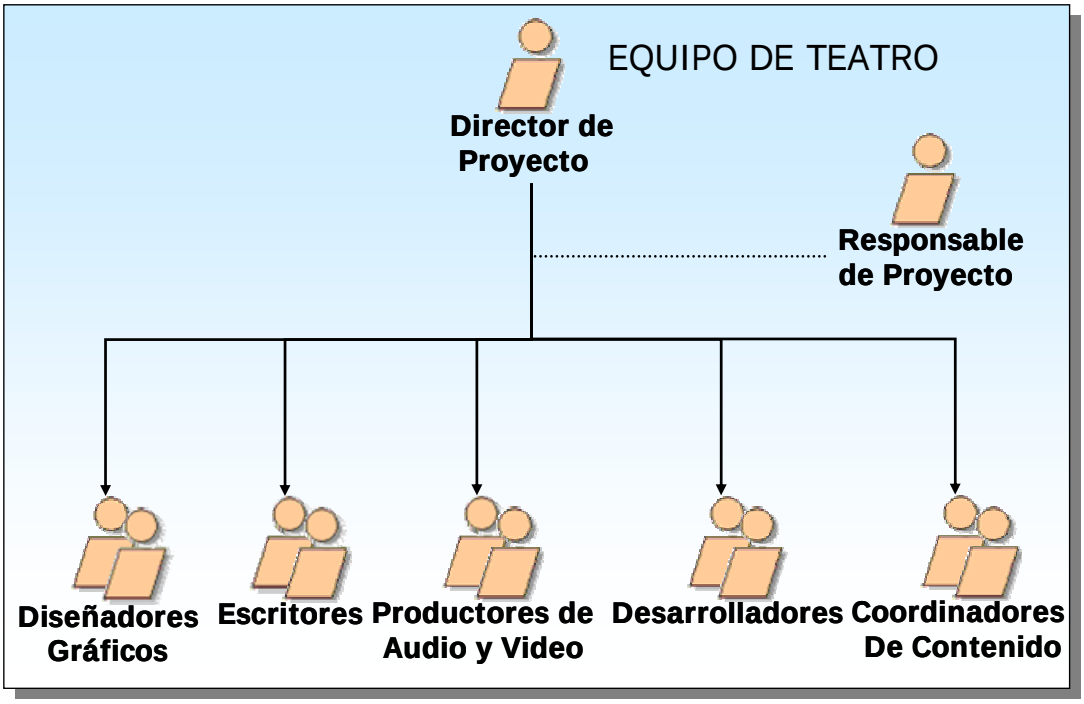

Figura 34: Estructura Organizacional del Equipo de Teatro

Para concluir, la siguiente tabla (adaptada de McConell 2003) resume las motivaciones de cada tipo de configuración de equipo, así como su clasificación

respecto a los objetivos que persiguen (resolución de problemas, creatividad o ejecución táctica):

	OBJETIVO GENERAL		
	Resolución de Problemas	**Creatividad**	**Ejecución táctica**
Característica	• Confianza	• Autonomía	• Claridad
Ejemplo	• Mantenimiento • Soporte	• Desarrollo de nuevos productos	• Actualización de un producto
Énfasis en el proceso	• Centrado en cuestiones	• Explorar posibilidades	• Tareas centradas • Funciones claras
Modelo apropiado de Desarrollo	• Codificar-Corregir • Espiral • Incremental	• Prototipado. • Espiral. • Incremental.	• Cascada. • Espiral. • Incremental.
El personal para este Equipo debe ser:	• Inteligente • Desenvuelto • Sensible • Integro	• Cerebral. • Independiente • Pensador. • Con Iniciativa. • Tenaz.	• Leal. • Comprometido. • De Acción. • Diligente. • Responsable.
Configuración Apropiada de Equipo	• Negocios • Búsqueda-Rescate • SWAT	• Negocios • Programador Jefe • En la sombra • Prestaciones • Equipo de teatro	• Negocios. • Programador Jefe • Prestaciones • SWAT • De Especialistas

Tabla 15: Resumen Configuraciones de Equipo y sus Características

Según la Tabla 15, es evidente que los procesos de desarrollo de software que consideran a este como un producto evolutivo, se adaptan mejor a las motivaciones y configuraciones de todos los equipos de desarrollo. Por esta razón, **Agile-DISOP**, re-toma este mismo espíritu de desarrollo con el propósito ser adaptable a cualquier situación. Por otra parte, es claro que con tantos miembros[63], una comunidad de desarrollo, encuentre difícil concebir una configuración única de equipo que atienda sus necesidades básicas (auto-gestión, comunicación y coordinación) así como las necesidades de los proyectos en ejecución (gestión y desarrollo). En el siguiente

[63] Con una alta heterogeneidad socio-técnica y motivacional.

numeral, se propone una estrategia para la organización del recurso humano de la comunidad de desarrollo.

5.8.1.9. Configuración del Equipo en Comunidad

Las comunidades de desarrollo plantean problemas especiales de comunicación y de coordinación motivados por el número de sus miembros y la dispersión geográfica entre ellos. Si un proyecto solo lo ejecuta una persona, entonces esta puede trabajar como quiera, ya que no tiene que comunicarse o coordinarse con nadie. Sin embargo, a medida que se incrementa el número de participantes en un proyecto, aumenta también de forma exponencial el número de canales de comunicación y la necesidad de coordinación entre ellos. La Figura 35, muestra este hecho. Los canales de comunicación crecen según la ecuación siguiente (asumiendo que todo el mundo habla con todo el mundo):

- **CC** = $((NM)^2 - NM)/2$. Donde **NM** es el Número de Miembros de la Comunidad.

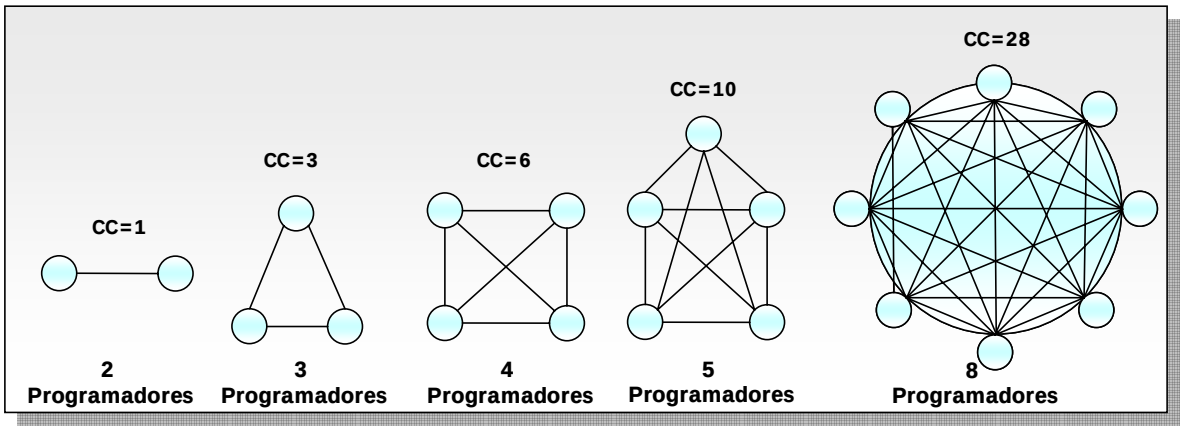

Figura 35: Vías de Comunicación en Proyectos de Distintos Tamaños

Cuanto más alto el **CC** para una comunidad de desarrollo, más tiempo y esfuerzo se destinara a las comunicaciones entre sus miembros, incrementando la posibilidad

de errores en la misma. Sin embargo, aunque en una comunidad de desarrollo que este soportada por una plataforma de integración (descrita en el numeral 5.15), se facilite establecer canales de comunicación entre todos sus miembros, es recomendable concebir una estructura organizacional donde existan personas que se comporten como interfaces entre equipos con el fin de lograr una organización y formalización de las comunicaciones de la comunidad. Para lograr esto, se requiere estructurar algún tipo de jerarquía creando equipos más compactos y al mismo tiempo, asignar responsabilidades relacionadas con la comunicación y coordinación con el propósito de alcanzar la interacción entre estos equipos y a través de la plataforma de integración.

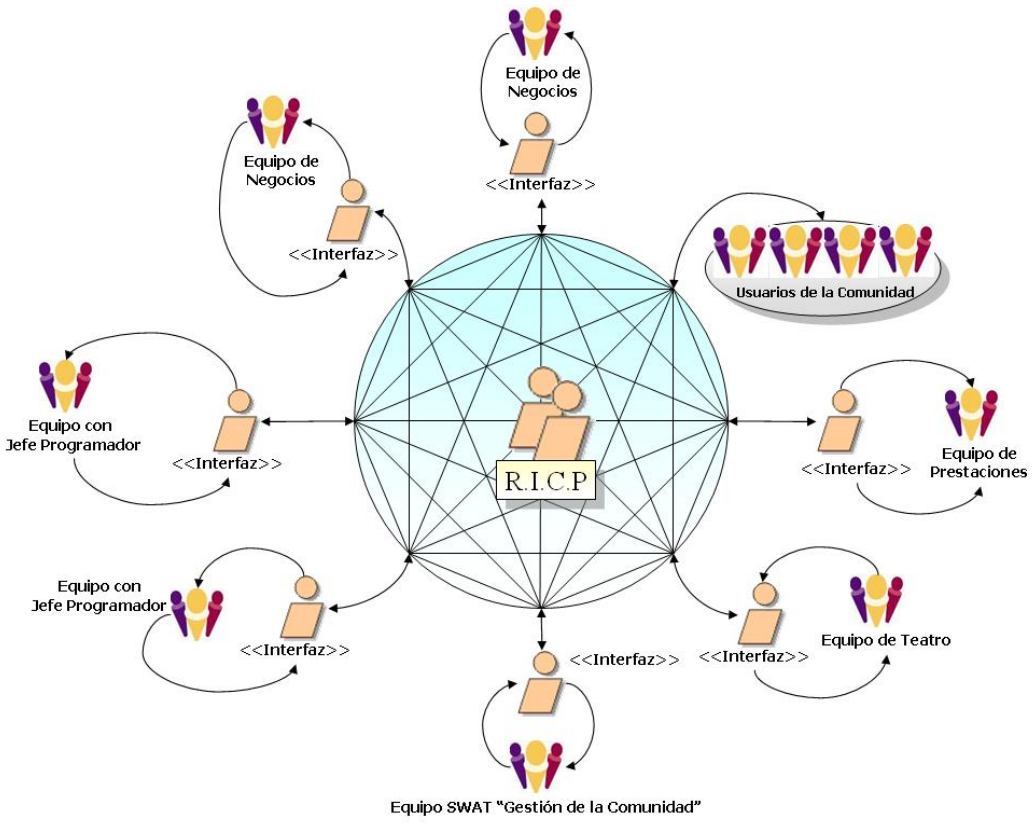

Figura 36: Organización de los equipos en la comunidad de desarrollo

La Figura 36, ilustra esta estrategia, siguiendo el consejo de McConell (2003) se adopta una estructura organizacional para la comunidad de desarrollo que acople equipos geográficamente dispersos mediante interfaces humano-computador que son responsables de la garantía, mantenimiento y soporte de los canales de comunicación con el propósito de asegurar una coordinación limpia y eficaz al interior de la comunidad. Por otra parte, e independientemente de la organización de los equipos pequeños, resulta crítico que haya una persona o equipo como responsable final de la integridad conceptual del producto. Su trabajo consiste en asegurarse de que todas las buenas soluciones locales de los equipos (i+d) conforman una buena solución global (McConell 2003). En la Figura 36, esta persona o equipo se denota con la sigla **R.I.C.P** acrónimo de: *"Responsable de la Integridad Conceptual del Producto"*. Como se indicara mas adelante en este mismo capitulo, **R.I.C.P** es desempeñado principalmente por el equipo promotor del proyecto.

Por otra parte, la Figura 36, ilustra la posibilidad de implementar varias configuraciones de equipo dependiendo de las motivaciones de la comunidad, que pueden incluir: resolución de problemas, creatividad y ejecución táctica. En consecuencia en esta figura se observan configuraciones de equipo que obedecen a alguna de estas tres motivaciones, sin embargo cada equipo cuenta con una interfaz de comunicación y coordinación con los demás. Finalmente, todo miembro de un equipo cualquiera, también es miembro de la comunidad y como tal, tendrá disponibilidad de servicios TIC´s de la plataforma de integración para comunicarse y coordinarse como miembro individual con cualquier otro.

Finalmente, la Figura 36, muestra un equipo dedicado a la gestión de la comunidad misma. Este equipo, debe ser experto en los temas relacionados con la plataforma de integración de la comunidad, así como en el proceso de desarrollo **Agile-DISOP** propuesto, con el propósito de centrarse en la resolución de los problemas cotidianos de la comunidad relativos a la plataforma de integración o la adopción del proceso. Se sugiere implementar una configuración **SWAT** (descrito en el numeral

5.8.1.6) para el equipo que será responsable de la gestión de la comunidad (descrita como disciplina en el numeral 5.10).

Una vez expuestas las posibilidades en lo relacionado con las configuraciones de equipo para la comunidad (i+d), a continuación, se especifican las disciplinas de la metodología **Agile-DISOP,** según la secuencia y descriptores mostrados en la Figura 24.

5.9. DISCIPLINA: GESTION DEL ENTORNO DE AGILE-DISOP

5.9.1. Propósito

El propósito fundamental de esta disciplina consiste en animar a la comunidad de desarrollo o cualquier equipo (i+d) a configurar el proceso y la plataforma de integración para cada proyecto en particular. Esta disciplina afirma la práctica propuesta en el numeral 5.4.9 y al mismo tiempo guarda estrecha relación con la disciplina propuesta en el numeral 5.13.

5.9.2. Conceptos

Los conceptos involucrados en esta disciplina son: Entorno, Plataforma de Integración, Comunidad de Desarrollo, Proyecto, Ingeniero de Procesos, Proceso de Desarrollo, Instanciación del Proceso de Desarrollo, Infraestructura de Desarrollo, Equipo de Soporte Técnico, Servicio de Help-Desk. Para encontrar las definiciones correspondientes a los anteriores conceptos favor referirse al Anexo 3.

5.9.3. Flujo de Trabajo

Como puede apreciarse en la Figura 37: Flujo de Trabajo para la Gestión del Entorno, en las iteraciones tempranas de la fase de “Inicio”, se prepara el entorno y

la plataforma de integración el proyecto. Las actividades involucradas en este flujo de trabajo, producirán un proceso de desarrollo específico para el proyecto.

Luego, durante las iteraciones más tardías, se hace necesario ir ajustando tanto el entorno del proyecto como la plataforma de integración, con el propósito de adaptar las variaciones imprevistas que surjan durante la ejecución del proyecto.

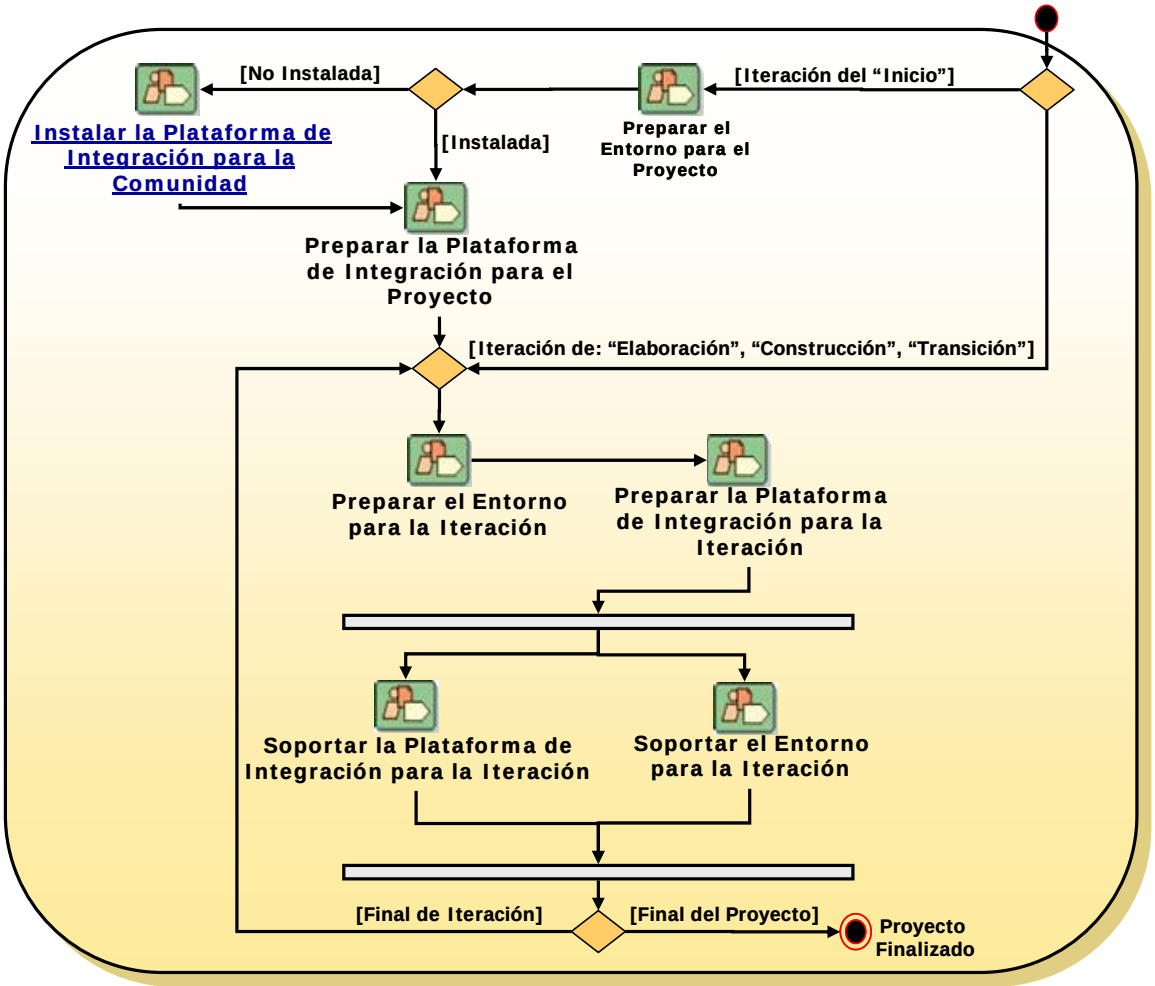

Figura 37: Flujo de Trabajo para la Gestión del Entorno

5.9.4. Actividades

A continuación se especifican las actividades que se muestran en la Figura 37: Flujo de Trabajo para la Gestión del Entorno.

Nombre de Actividad	Preparar el Entorno para el Proyecto
Descripción	El propósito de esta actividad consiste en transformar el proceso **Agile-DISOP** genérico en uno específico para el proyecto, dependiendo del nivel de habilidad, conocimiento y experiencia del equipo de desarrollo. Además, esta transformación tiene que ver con el nivel de rigurosidad y madurez con el que se desea implementar el proceso, pues según lo recomendado en el numeral 5.5, la adopción del proceso debe ser iterativa e incremental para no sofocar a los equipos (i+d) con poca experiencia. De otra parte esta actividad está en concordancia con la disciplina explicada en el numeral 5.13 sobre el mejoramiento también iterativo e incremental del proceso durante su adopción y uso con el propósito de alcanzar madurez. Preparar un proceso específico para el proyecto involucra: ▪ Determinar las desviaciones y/o extensiones que se van a realizar al proceso en el marco del desarrollo del proyecto. ▪ Seleccionar los artefactos del proceso que se van a usar en el proyecto y estableciendo para ellos criterios de calidad con el fin de valorarlos. ▪ Preparar lineamientos y plantillas estándar específicos para el proyecto y la naturaleza del producto que se va a desarrollar. ▪ Producir una lista de tecnologías de software candidatas para el desarrollo.
Entradas	▪ **Agile-DISOP** (Genérico) **Ver Anexo 5**
Salidas	▪ **Agile-DISOP** (Específico para el Proyecto)
Roles Responsables:	▪ Ingeniero de Procesos. (RUP 2003)
Nombre de Actividad	Instalar la Plataforma de Integración para la Comunidad
Descripción	Si aún no está instalada la plataforma de servicios TIC que integrará a la comunidad de desarrollo, entonces debe hacerse cuanto antes con el propósito de soportar las áreas de la gestión, comunicación coordinación y control de los equipos (i+d) durante el desarrollo del proyecto.
Entradas	Instaladores de la Plataforma de Integración, Equipo Servidor y Plataforma SW
Salidas	Plataforma de Integración Instalada
Roles Responsables:	Ingeniero de Procesos. (RUP 2003)
Nombre de Actividad	Preparar la Plataforma de Integración para el Proyecto
Descripción	Consiste en configurar y definir la información general del proyecto en términos de tiempo, personas, responsabilidades y productos, mediante el uso de los servicios ofrecidos por la plataforma de integración que usa la comunidad de desarrollo.
Entradas	▪ **Plataforma de Integración** (Sin información específica para el proyecto) ▪ **Plan de Proyecto** (Ver Anexo 11)
Salidas	▪ **Plataforma de Integración** (Con información específica para el proyecto)
Roles Responsables:	▪ Gestor del Proyecto. (RUP 2003)
Nombre de Actividad	Preparar el Entorno para la Iteración

Descripción	El propósito de esta actividad es similar al de la actividad **"Preparar el Entorno para el Proyecto"**, la única diferencia es que se va adaptando para cada iteración en particular.
Entradas	▪ **Agile-DISOP** (Específico para el Proyecto) ▪ **Plan de Iteración** (Ver Anexo 5)
Salidas	▪ **Agile-DISOP** (Específico para la Iteración)
Roles Responsables:	▪ Ingeniero de Procesos. (RUP 2003)
Nombre de Actividad	Preparar la Plataforma de Integración para la Iteración
Descripción	Consiste en configurar y definir la información general del proyecto en términos de tiempo, personas, responsabilidades, productos entre otros, que son específicos para una iteración del proyecto mediante el uso de los servicios ofrecidos por la plataforma de integración que usa la comunidad de desarrollo.
Entradas	▪ **Plataforma de Integración** (Sin información específica para la Iteración) ▪ **Plan de Iteración** (Ver Anexo 5)
Salidas	▪ **Plataforma de Integración** (Con información específica para la Iteración)
Roles Responsables	▪ Gestor del Proyecto. (RUP 2003)
Nombre de Actividad	Soportar la Plataforma de Integración para la Iteración
Descripción	El propósito de esta actividad consiste en soportar al equipo (i+d) en el uso de los servicios informáticos que provee la plataforma integración durante una iteración específica. La idea es que el equipo (i+d) no se disperse en peculiaridades técnicas relacionadas con el funcionamiento de la plataforma. En cualquier caso, el equipo (i+d) realizará peticiones de soporte al equipo de soporte técnico el cuál las recopilará y evaluará para darles respuesta tan pronto como sea posible.
Entradas	▪ **Servicio de "Help-Desk"** de la Plataforma de Integración seleccionada.
Salidas	▪ **Servicio de "Help-Desk"** de la Plataforma de Integración seleccionada.
Roles Responsables	▪ Ingeniero de Procesos. (RUP 2003) ▪ Equipo (i+d)
Nombre de Actividad	Soportar el Entorno para la Iteración
Descripción	El propósito de esta actividad consiste en soportar al equipo (i+d) en el uso de las herramientas y el proceso mismo para una iteración específica. Es crucial para el desempeño del equipo (i+d) y la velocidad-eficacia de la ejecución del proyecto el soporte relacionado con los problemas que se puedan presentar a cada momento. Incluye la instalación del software requerido (utilidades) para garantizar que el hardware funciona como es debido y que el potencial de transmisión de la red no es perturbado.
Entradas	**Servicio de "Help-Desk"** de la Plataforma de Integración seleccionada.
Salidas	**Servicio de "Help-Desk"** de la Plataforma de Integración seleccionada.
Roles Responsables	▪ Administrador del Sistema. (RUP 2003) ▪ Equipo (i+d)

Tabla 16: Actividades de la Gestión del Entorno

5.9.5. Artefactos

Los artefactos que se elaboran, refinan y/o utilizan durante la ejecución de las actividades de esta disciplina se relacionan en la siguiente tabla:

No.	Nombre	Propósito	Especificado en…
1	**Agile-DISOP**	Consiste en el modelo del proceso de desarrollo, el cuál puede personalizarse para diferentes proyectos, equipos y nivel de experiencia.	Capitulo 5.
2	**Plataforma de Integración**	Consiste en la plataforma software de integración para la comunidad de desarrollo, la cual provee servicios de valor agregado para la gestión, comunicación y coordinación de los equipos (i+d) bajo un ambiente de dispersión geográfica.	Numeral 5.15.
3	**Plan de Proyecto**	Este artefacto hace parte de la disciplina de gestión del proyecto (véase numeral 5.11). en este documento reúne las estimaciones acerca del esfuerzo, presupuesto, personal, problema y objetivos del proyecto.	Anexo 11
4	**Plan de Iteración**	Este artefacto hace parte de la disciplina de gestión del proyecto (véase numeral 5.11). En este documento se reúnen preliminarmente las estimaciones acerca del esfuerzo, presupuesto, personal, problema y objetivos de la iteración.	Anexo 5

Tabla 17: Artefactos de la Disciplina Gestión del Entorno

5.9.6. Arquitectura

Como se muestra en la Figura 38, el proceso **Agile-DISOP** genérico es personalizado para el proyecto o proyectos que emprenda la comunidad de desarrollo. Así en un momento determinado pueden existir varias “instancias” del mismo proceso **Agile-DISOP** ejecutándose en diferentes sub-proyectos con un nivel variado de madurez y dentro un macro-proyecto de desarrollo en comunidad. De otra parte, se puede observar que cada proyecto emprendido por la comunidad de

desarrollo, será soportado mediante una plataforma de integración que usa la comunidad y que se describe suficientemente en el numeral 5.15.

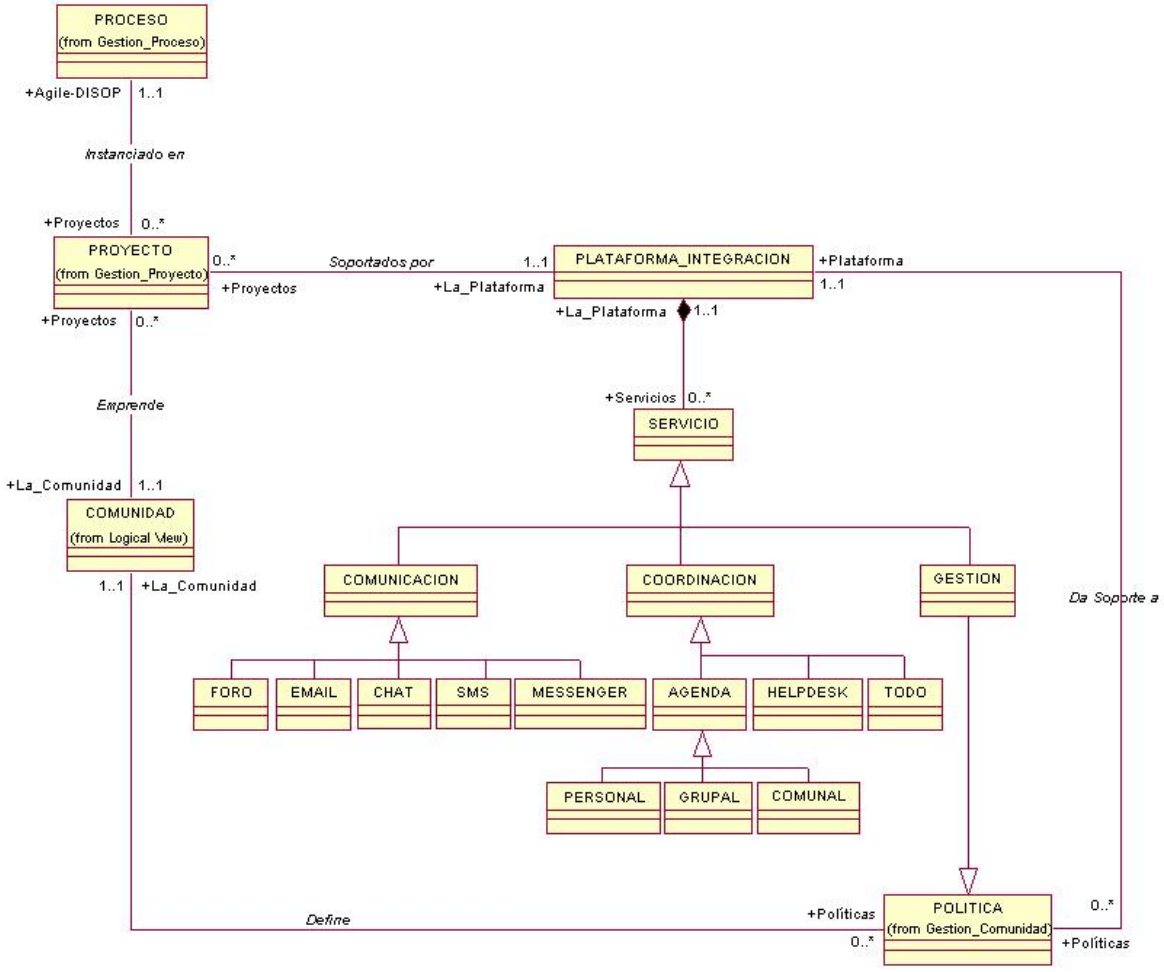

Figura 38: Arquitectura de la Gestión del Entorno

5.9.7. Premisas

- La plataforma de integración debe ser accesible por todos y cada uno de los miembros de la comunidad, desde cualquier ubicación geográfica, salvo casos de fuerza mayor.

- De otra parte la guía del proceso **Agile-DISOP,** debe estar publicada en la plataforma de integración u otra plataforma de gestión de contenidos (se recomienda Moodle [8]) con el objeto de facilitar la referencia y consulta por parte de los miembros de la comunidad de desarrollo.

- Debe existir una configuración de equipo de búsqueda y rescate (explicado en el numeral 5.8.1.5) disponible para la resolución ágil de problemas que puedan surgir con la instalación, puesta en marcha y mantenimiento de la plataforma ante toda la comunidad de desarrollo.

- Es fundamental que exista un modelo de documentación que permita instrumentar esta disciplina. Se recomienda la estructura de directorios del Anexo 5. Allí se muestra la ubicación exacta de los artefactos de esta disciplina en el marco de una iteración para cualquier fase del proceso. La idea es que con cada nueva iteración, el usuario de Agile-DISOP, sólo replicará el sub-directorio Iteración[64], logrando con esto, alcanzar una estructura de documentación uniforme para todos los proyectos que emprenda la comunidad.

5.10. DISCIPLINA: GESTION DE LA COMUNIDAD DE AGILE-DISOP

5.10.1. Propósito

El propósito fundamental de esta disciplina consiste en asegurar la gestión efectiva de la comunidad de desarrollo mediante la puesta en marcha de mecanismos adecuados para favorecer la sociabilidad de los equipos (i+d) bajo un ambiente de dispersión geográfica. Algunos de estos mecanismos estarán soportados por servicios informáticos publicados en la plataforma de integración recomendada por

[64] Asignándole el número de iteración al directorio recién creado.

la práctica del numeral 5.4.9 y descrita en el numeral 5.15. Según Preece (2003), la sociabilidad de una comunidad de desarrollo enfatiza en la interacción social. Las comunidades con una buena sociabilidad establecen políticas que dan soporte al propósito mismo de la comunidad, son entendidas y aceptadas por todos los miembros y finalmente son lo suficientemente prácticas como para ser implementadas con facilidad. La sociabilidad evoluciona con la comunidad, pero siempre enfatizara en el favorecimiento de su propósito.

5.10.2. Conceptos

Los conceptos involucrados en esta disciplina son: Sociabilidad y Políticas Para encontrar las definiciones correspondientes a los anteriores conceptos favor referirse al Anexo 3.

5.10.3. Flujo de Trabajo

Como puede observarse en la Figura 39, la gestión de la comunidad consiste en varias actividades simultáneas que son responsables por distintos aspectos relativos al comportamiento y la auto-regulación de la comunidad. La gestión de la comunidad es responsabilidad del equipo SWAT mostrado en la Figura 36 y descrito en el numeral 5.8.1.6.

La gestión de la comunidad propone un marco para regular lo que la gente puede/no puede, debe/no debe hacer. Esta función es crítica, puesto que de la calidad de su ejecución depende en gran medida el crecimiento o decadencia de la misma. Si la gestión es débil, entonces la comunidad sucumbirá bajo la anarquía; demasiada fuerza en la gestión, hará parecer a la comunidad como un centro correccional donde nadie se siente a gusto. Como se observa en la Figura 39, una vez se ha estabilizado la definición del propósito de la comunidad, la gestión de la comunidad se divide en dos grandes áreas:

- **La gestión de miembros**. Constituida por las actividades relacionadas con la gestión del acceso a la comunidad, la gestión de roles y la gestión de la comunicación efectiva entre los miembros de la comunidad de desarrollo.

- **La Gestión de Políticas**. Constituida por las actividades relacionadas con la gestión del registro de los miembros en la comunidad, la gestión de las políticas de comportamiento y la gestión de la confianza-seguridad a través de la comunidad.

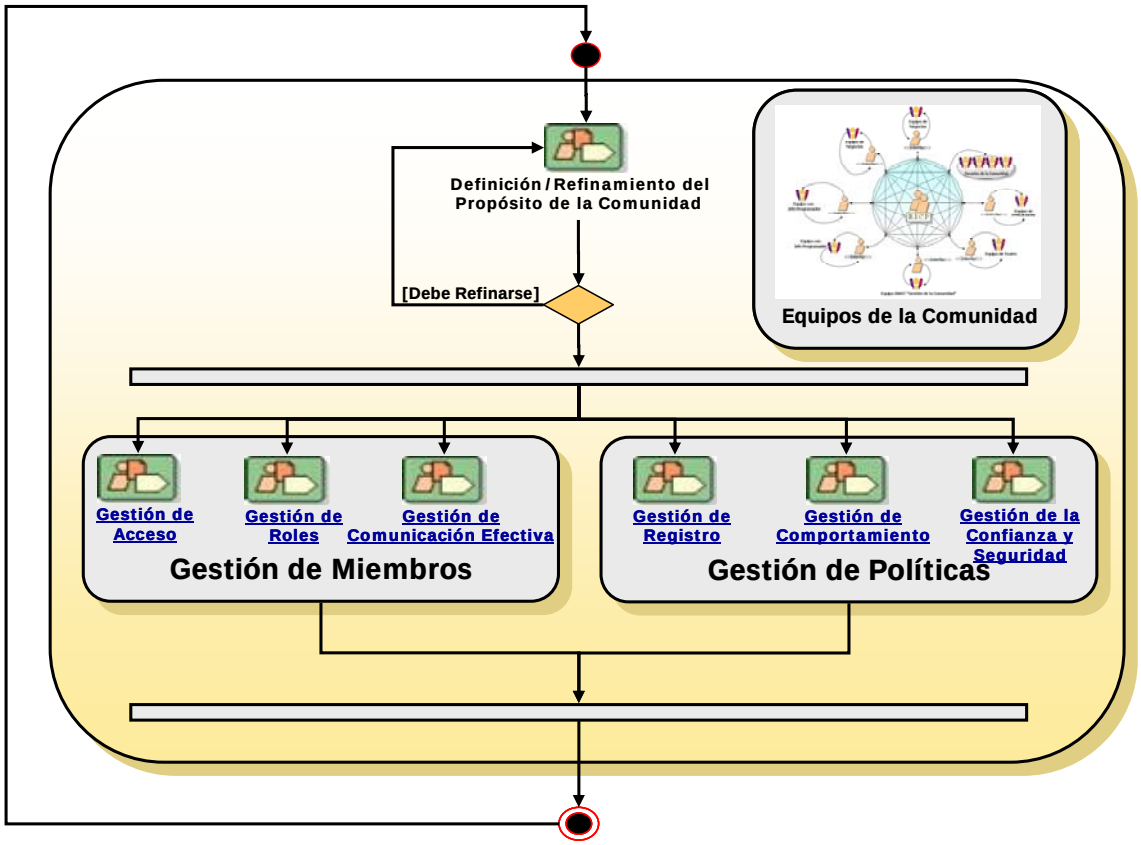

Figura 39: Flujo de Trabajo de la Gestión de la Comunidad

5.10.4. Actividades

A continuación se especifican las actividades que se muestran en la Figura 39.

Nombre de Actividad	Gestión del Acceso
Descripción	El propósito de esta actividad consiste en determinar y gestionar los requerimientos para unirse o dejar la comunidad de desarrollo. De la misma forma, es importante asegurar que los candidatos a miembros de la comunidad conocen estos requerimientos y los aceptan o rechazan. **Políticas de Adhesión.** El interesado en formar parte de la comunidad debe: ▪ Proporcionar un nombre de usuario y una contraseña. ▪ Proporcionar un nombre de cuenta de e-mail activada. ▪ Diligenciar la información personal del usuario especificado en el formulario del registro de la Figura 40. ▪ Esperar el estudio y aprobación de la solicitud de suscripción por parte del equipo de gestión de la comunidad. **Políticas de Abandono** ▪ El miembro de la comunidad debe expresar de forma explícita y durante una sesión activa, que desea retirarse de la comunidad de desarrollo. **Requerimientos Técnicos** • Acceso a Internet, Navegador de Internet, Ancho de Banda.
Entradas	▪ **Políticas de adhesión y/o abandono** (Versión i)
Salidas	▪ **Políticas de adhesión y/o abandono** (Versión i+1)
Roles Responsables:	▪ Equipo SWAT de Gestión de la Comunidad (mostrado en la Figura 36)
Nombre de Actividad	Gestión de Roles
Descripción	El propósito de esta actividad consiste en la gestión de los diferentes roles que los miembros de la comunidad de desarrollo van asumiendo dentro de la ejecución de los proyectos o durante la gestión de la comunidad misma. Existen dos (2) tipos de roles: visibles e invisibles. Los primeros hacen parte de los equipos (i+d) durante los proyectos de desarrollo (Ej. analistas, diseñadores, desarrolladores, etc.), los segundos (Ej. administradores, moderadores, personal de soporte) hacen parte del equipo de gestión de la comunidad. Es crucial que la declaración de responsabilidades asociadas a cada rol sea clara para todo miembro en la comunidad de desarrollo con el propósito de evitar malentendidos. De otra parte, es importante tener en cuenta que la contribución individual de cada rol, debe favorecer el propósito de la comunidad de desarrollo.
Entradas	▪ **Políticas de Gestión de Roles en la Comunidad** (Versión i)
Salidas	▪ **Políticas de Gestión de Roles en la Comunidad** (Versión i+1)
Roles Responsables:	▪ Equipo SWAT de Gestión de la Comunidad (mostrado en la Figura 36)
Nombre de Actividad	Gestión de la Comunicación Efectiva
Descripción	Esta actividad tiene como propósito la determinación y gestión de las formas correctas de expresión en línea (íconos gestuales, jerga) que usan los miembros de la comunidad para comunicarse entre sí. Algunas reglas de cortesía pueden ser: • Evitar el envío de un mensaje que es relevante para una sola persona a toda una lista de miembros en la comunidad. • Responder con palabras como “Si”, “No”, sin contexto de la conversación, será incomprensibles en pocos días. Siempre debe

<table>
<tr><td></td><td>proporcionarse un contexto a las respuestas. Ejemplo: incluyendo el texto del mensaje al cuál se responde o una oración completa que sea comprensible.
• No publicar mensajes de poca relevancia en los espacios de la comunidad.
• Evitar las formas de humor que puedan ser malentendidas o potencialmente ofensivas.
• Evitar la redacción de mensajes en letras capitales, la gente lo interpretará como si UD. estuviese gritando.
• Evitar el uso de auto-respuestas cuando se esté suscrito a una lista.
• Agregue íconos gestuales (emoticons) a sus mensajes para incrementar las posibilidades de una interpretación correcta.
• Algunos ejemplos de íconos gestuales se ilustran a continuación:
 

Comunicación Efectiva con Moderación

• Así mismo, puede proveerse un mecanismo de mediación entre los miembros de la comunidad que inician un debate en cualquier punto dentro de la ejecución del proyecto de desarrollo. Por ejemplo: un debate puede crearse para negociar los requerimientos, recursos y alcances de un producto software en desarrollo, en este caso es importante contar con un mecanismo de moderación que mantenga el debate dentro del objetivo de la negociación. La función de moderación también puede tomar medidas drásticas en caso de que el código de comportamiento sea violentado por alguno de los miembros de la comunidad dentro de un debate o negociación.</td></tr>
<tr><td>Entradas</td><td>▪ Políticas de Netiquette (Versión i)</td></tr>
<tr><td>Salidas</td><td>▪ Políticas de Netiquette (Versión i+1)</td></tr>
<tr><td>Roles Responsables:</td><td>▪ Equipo SWAT de Gestión de la Comunidad (mostrado en la Figura 36)</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>Nombre de Actividad</td><td>Gestión del Registro</td></tr>
<tr><td>Descripción</td><td>El propósito fundamental de esta actividad consiste en regular las políticas de registro que garantizan un mecanismo de moderación en lo relativo a las personas que pueden unirse a la comunidad. Las políticas de registro pueden concebirse para filtrar determinadas tendencias, capacidades o grupos de personas que podrían no contribuir o incluso deteriorar el clima dentro de la comunidad de desarrollo. Las políticas de registro pueden determinar las condiciones para que usuarios potenciales de la comunidad puedan unirse durante un tiempo limitado con carácter de invitado.</td></tr>
</table>

Entradas	▪ **Políticas de Registro** (Versión i)
Salidas	▪ **Políticas de Registro** (Versión i+1)
Roles Responsables:	▪ Equipo SWAT de Gestión de la Comunidad (mostrado en la Figura 36)
Nombre de Actividad	Gestión del Comportamiento
Descripción	El propósito fundamental de esta actividad consiste en establecer y gestionar un código de comportamiento aceptable para todos los miembros de la comunidad de desarrollo, que garantice una relación cordial y responsable entre los miembros al mismo tiempo que permite a la comunidad crecer y evolucionar. Se proponen las siguientes directivas para empezar: • Las expresiones o palabras son responsabilidad del usuario y no expresan el sentir de la comunidad ni de sus directivos. • La publicación de información ilegal, flageada o inmoral no está permitida. • El trato agresivo no está permitido.
Entradas	**Código de Comportamiento** (Versión i)
Salidas	**Código de Comportamiento** (Versión i+1)
Roles Responsables:	▪ Equipo SWAT de Gestión de la Comunidad (mostrado en la Figura 36)
Nombre de Actividad	Gestión de la Confianza y Seguridad
Descripción	La confianza es crucial en la buena salud de una comunidad. El propósito de esta actividad consiste en asegurar la protección sobre la privacidad y confidencialidad de cada miembro de la comunidad de desarrollo. Preece (2003) recomienda los siguientes lineamientos: • Protección de la información confidencial. • Protección de información financiera. • Advertir sobre información cuyo uso incorrecto puede causar problemas. • Crear mecanismos para la protección de la propiedad intelectual. • Instalar mecanismos que apoyen la confianza. Ejemplo: logos de las universidades y equipos (i+d) involucrados en la comunidad, fotografías e información veraz para todos los miembros etc.
Entradas	▪ **Políticas de Gestión de la Confianza y Seguridad** (Versión i)
Salidas	▪ **Políticas de Gestión de la Confianza y Seguridad** (Versión i+1)
Roles Responsables	▪ Equipo SWAT de Gestión de la Comunidad (mostrado en la Figura 36)

Tabla 18: Actividades del Flujo de Gestión de la Comunidad

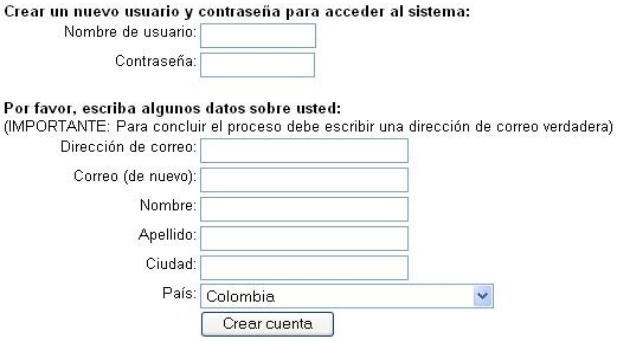

Figura 40: Formulario de Registro para Nuevos Usuarios

5.10.5. Artefactos

Los artefactos elaborados y/o refinados durante la ejecución de la gestión de la comunidad, se describen en la siguiente tabla:

No.	Nombre	Propósito	Especificado en…
1	Políticas de Adhesión y/o Abandono.	Especifica las restricciones para la aceptación o rechazo del aspirante a miembro de la comunidad.	Anexo 5
2	Políticas de Gestión de Roles de la Comunidad.	Especifica las responsabilidades de cada rol en la comunidad.	Anexo 5
3	Reglas de Netiquette	Especifica el protocolo aceptable de expresión y comunicación de los miembros de la comunidad.	Anexo 5
4	Políticas de Registro	Especifica los requerimientos técnicos y de información que son exigidos durante el proceso de suscripción a la comunidad de desarrollo.	Anexo 5
5	Código de Comportamiento	Especifica el comportamiento aceptable de cada miembro de la comunidad.	Anexo 5
6	Políticas de Gestión de Confianza y Seguridad	Especifica el esquema de promoción al reconocimiento de la propiedad intelectual y la protección de la información personal.	Anexo 5

5.10.6. Arquitectura

Como se muestra en la Figura 41, la arquitectura de la gestión de la comunidad involucra la jerarquía de políticas que regulan el comportamiento de la comunidad con ánimo de satisfacer su propósito fundamental. Las políticas de gestión de la comunidad deben estar soportadas por los servicios de la plataforma de integración y serán gestionadas por el equipo SWAT descrito en el numeral 5.8.1.6 .

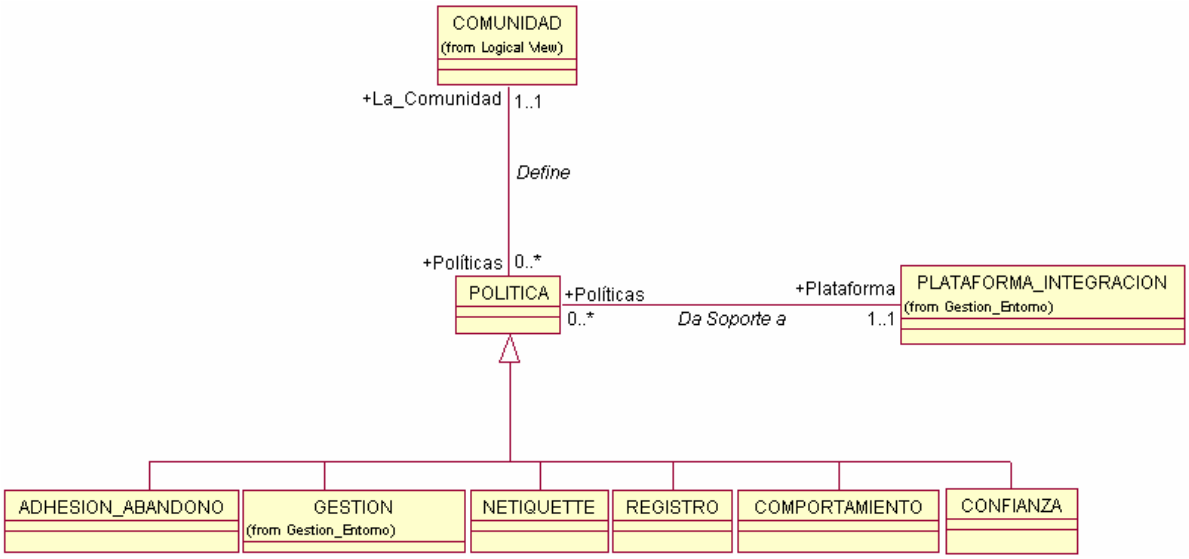

Figura 41: Arquitectura de la Gestión de la Comunidad

5.10.7. Premisas

- Debe existir un proceso continuo de refinamiento y puesta en marcha sobre las políticas de la comunidad que además esté regulado por una vigilancia eficaz sobre el cumplimiento de las mismas.

- Las políticas deben ser aceptadas, públicas, consensuadas y claras para todos los miembros de la comunidad de desarrollo, en ningún caso deben promover la discriminación sobre etnias o grupos sociales específicos, sin embargo un mecanismo de filtro puede implementarse con base en los criterios de: conocimiento, expectativas y contribuciones por parte de los usuarios que aspiran a convertirse en miembros activos de la comunidad.

- Las políticas deben ser fáciles de implementar en la plataforma de integración con los servicios que esta provee.

- Debe existir una configuración de teatro (explicado en el numeral 5.8.1.8) disponible para la elaboración del modelo de sociabilidad ante toda la comunidad de desarrollo.

- Es fundamental que exista un modelo de documentación que permita instrumentar esta disciplina. Se recomienda la estructura de directorios del Anexo 5. Allí se muestra la ubicación exacta de los artefactos de esta disciplina en el marco de una iteración para cualquier fase del proceso. La idea es que con cada nueva iteración, el usuario de Agile-DISOP, sólo replicará el sub-directorio Iteración[65], logrando con esto, alcanzar una estructura de documentación uniforme para todos los proyectos que emprenda la comunidad.

5.11. DISCIPLINA: GESTIÓN DEL PROYECTO PARA AGILE-DISOP

5.11.1. Propósito

El objeto fundamental de esta disciplina consiste en proveer un marco de referencia para la gestión al determinar unos criterios para la planificación, ejecución, monitoreo, control y manejo del riesgo en proyectos informáticos. La Gestión de Proyectos contempla elementos existentes en cualquier trabajo de dirección, pero al mismo tiempo, tiene en cuenta la interacción entre diversos sub-proyectos ejecutados por equipos dispersos geográficamente. La disciplina de gestión de **Agile-DISOP** se divide en dos:

- **Sub-Disciplina de Planificación.** Se concentra en determinar las características principales del macro-proyecto como son: Objetivos y alcance del software a desarrollar, costo aproximado, cronograma y recursos (humanos, tecnológicos, de infraestructura entre otros). Así mismo, el sub-

[65] Asignándole el número de iteración al directorio recién creado.

modelo de planificación promoverá la partición del Macro-Proyecto en Sub-proyectos más pequeños, los cuáles serían ejecutados por equipos (i+d) independientes y geográficamente dispersos. Lo anterior, es determinado durante la fase de **"Inicio"** del macro-proyecto, pero se refinaría constantemente de forma más realista a partir de la fase de "Elaboración" que es donde se realizan las primeras iteraciones sobre los casos de uso arriesgados en cualquier sub-proyecto.

- **Sub-Disciplina de Monitoreo, Control y Gestión de Riesgos.** Se concentra en proveer los mecanismos de acompañamiento a las actividades de la planificación que brindarán el soporte para realizar el seguimiento y evaluación del estado de avance del macro y micro proyectos dentro de los plazos y costos establecidos. Además, según Boehm (1989), la gestión de riesgos se divide en dos: estimación y control. La estimación se concentra en la identificación, análisis y priorización de los riesgos, mientras que el control se preocupa de la planificación de la gestión, resolución y monitorización de riesgos.

5.11.2. Conceptos

Los conceptos involucrados en esta disciplina son: Promotor del Proyecto, Director del Proyecto, Gestión, Monitoreo, Control, Riesgos, Cronograma, Recursos, Productos. Para encontrar las definiciones correspondientes a los anteriores conceptos favor referirse al Anexo 3.

5.11.3. Flujo de Trabajo

En la Figura 42: Flujo de Trabajo para la Gestión del Macro-Proyecto y la Tabla 19, se ilustra y describe la secuencia de actividades que se siguen desde la propuesta del macro-proyecto a la comunidad, pasando por su refinamiento y ajuste, hasta su particionamiento en varios sub-proyectos.

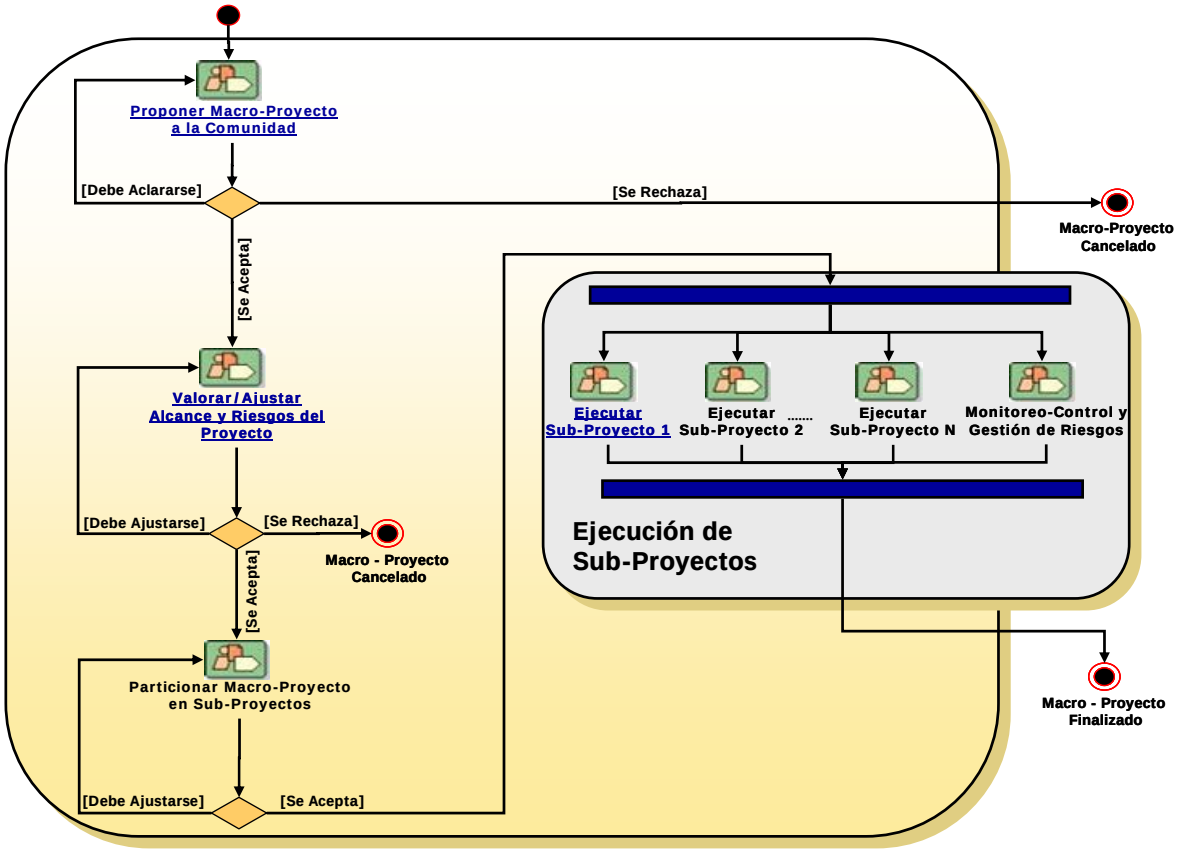

Figura 42: Flujo de Trabajo para la Gestión del Macro-Proyecto

Cada sub-proyecto se ejecuta siguiendo el flujo de trabajo que se muestra en la Figura 43. Obsérvese que el flujo de trabajo refleja la naturaleza ágil, iterativa e incremental y comunitaria del proceso recomendada en la práctica planteada en el numeral 5.4.3. Mas adelante, en la Tabla 20: Actividades del Flujo de Gestión a nivel de Sub-Proyecto se especifican todas las actividades de la Figura 43.

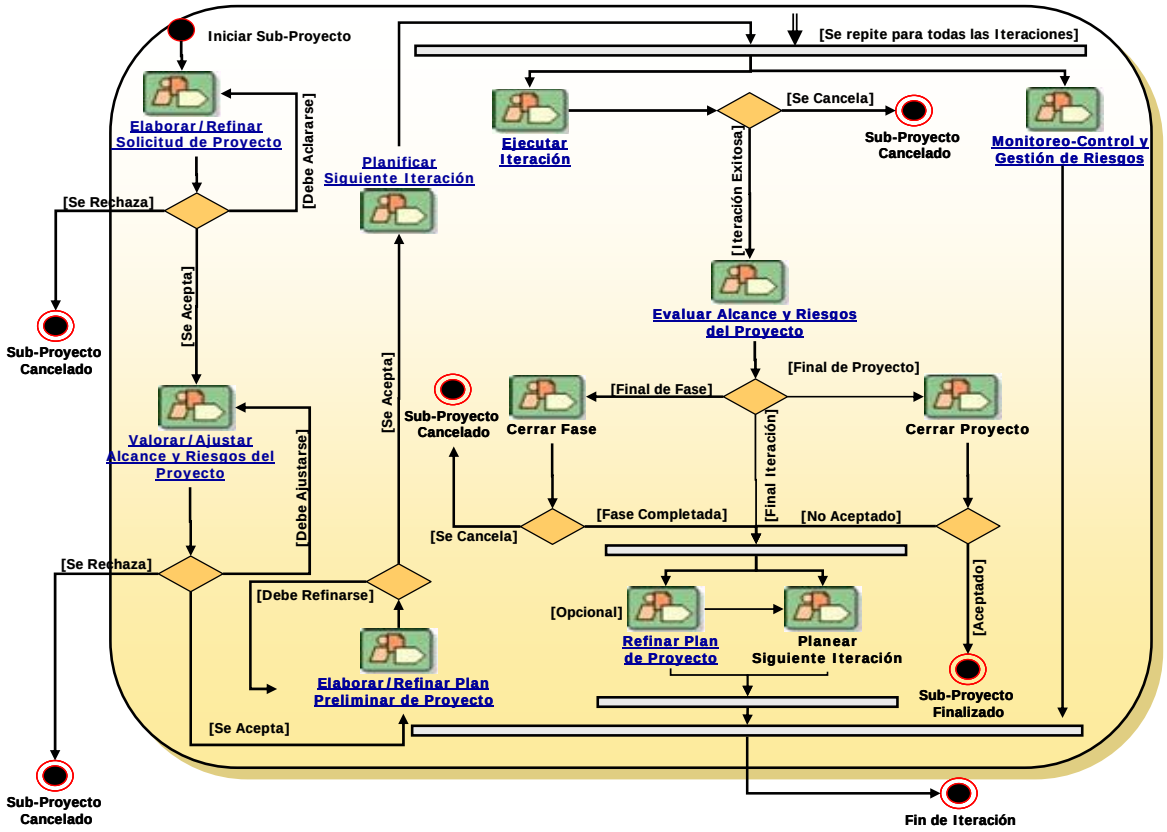

Figura 43: Flujo de Trabajo para el Modelo de Gestión en un Sub-Proyecto

5.11.4. Actividades

La descripción de las actividades de la Figura 42: Flujo de Trabajo para la Gestión del Macro-Proyecto se especifica a continuación:

Nombre de Actividad	Proponer el Macro-Proyecto a la Comunidad
Descripción	Consiste en la publicación ante la comunidad de desarrollo de una idea de software por parte cualquier equipo (i+d). La propuesta estará sometida a la aprobación de la comunidad para proseguir. El Anexo 4 especificada la información de esta propuesta en el artefacto "Visión del Proyecto". Se recomienda proponer una idea con potencial y ojalá acompañada de un prototipo preliminar, tal como plantea la práctica del numeral 5.4.3
Entradas	▪ **Visión de Macro-Proyecto** (En Blanco) Véase Anexo 4
Salidas	▪ **Visión de Macro-Proyecto** (Diligenciado) ▪ **Opcional:** Prototipo Preliminar.
Roles Responsables:	▪ Miembro de la comunidad o Sub-Equipo (i+d) (Promotor Proyecto)

Nombre de Actividad	Valorar/Ajustar Alcances y Riesgos del Proyecto.
Descripción	Consiste en la estimación del riesgo que implica el proyecto y el ajuste a los alcances del mismo. Así mismo, se transforma el modelo general de métricas a uno específico para el proyecto. Esta visión debe refinarse hasta ser aprobada plenamente por la comunidad de desarrollo.
Entradas	▪ **Alcances del Macro-Proyecto** (En Blanco) ▪ **Modelo Métricas del Macro-Proyecto**: **Producto** (Anexo 1) **Modelo de Métricas del Macro-Proyecto: Proceso** (Anexo 9) ▪ **Plan Preliminar de Riesgos del Macro-Proyecto** (Anexo 6)
Salidas	▪ **Alcances del Macro-Proyecto** (Diligenciado) ▪ **Modelo Métricas del Macro-Proyecto**: **Producto y Proceso** (Específico) ▪ **Plan Preliminar de Riesgos del Macro-Proyecto** (Diligenciado)
Roles Responsables:	▪ Promotor del Proyecto ▪ **Colaboran:** o Equipo (i+d) del Promotor del Proyecto o Comunidad de Desarrollo
Nombre de Actividad	Particionar Macro-Proyecto en Sub-Proyectos
Descripción	Consiste en la partición del macro-proyecto en unidades más compactas denominadas sub-proyectos, los cuáles son ejecutados por equipos (i+d) independientes geográficamente dispersos. La aprobación en comunidad de esta partición garantizará el inicio de la ejecución de los sub-proyectos distribuidos a través de la comunidad.
Entradas	▪ **Distribución del Macro-Proyecto en Sub-Proyectos** (En Blanco) Véase Anexo 11.
Salidas	▪ **Distribución del Macro-Proyecto en Sub-Proyectos** (Diligenciado)
Roles Responsables:	▪ Promotor del Proyecto ▪ **Colaboran:** o Equipo (i+d) del Promotor del Proyecto o Comunidad de Desarrollo
Nombre de Actividad	Ejecutar Sub-Proyecto **i**
Descripción	La especificación de esta actividad se encuentra en la Figura 43: Flujo de Trabajo para el Modelo de Gestión en un Sub-Proyecto.
Entradas	▪ Ninguna
Salidas	▪ **Sub-Proyecto Finalizado / Sub-Proyecto Cancelado**
Roles Responsables:	▪ Director de Sub-Proyecto **i** ▪ Equipo (i+d) responsable del Sub-Proyecto i
Nombre de Actividad	Monitoreo, Control y Gestión de Riesgos
Descripción	Las actividades relacionadas con el monitoreo, control y gestión de riesgos, se realizan tanto a nivel del Macro-Proyecto como a nivel de cada sub-proyecto.
Entradas	▪ **Alcances del Macro-Proyecto** (Diligenciado) ▪ **Modelo Métricas del Macro-Proyecto**: **Producto y Proceso** (Específico) ▪ **Plan Preliminar de Riesgos del Macro-Proyecto** (Diligenciado) ▪ **Valoración: Gestión de Riesgos, Monitoreo y Control (Anexo 10)**
Salidas	▪ **Valoración: Gestión de Riesgos, Monitoreo y Control** (Diligenciada)
Roles Responsables	▪ Promotor del Proyecto

Tabla 19: Actividades del Flujo de Gestión a nivel de Macro-Proyecto

De otra parte, la especificación de las actividades de la Figura 43: Flujo de Trabajo para el Modelo de Gestión en un Sub-Proyecto se especifican a continuación:

Nombre de Actividad	Elaborar/Refinar Visión de Proyecto
Descripción	El equipo (i+d) que manifiesta su interés a la comunidad sobre el desarrollo de un sub-proyecto, elabora un documento de solicitud de proyecto de acuerdo al Anexo 4. Este documento se publica ante la comunidad y estará sometido a su aprobación para proseguir la ejecución.
Entradas	▪ **Visión de Sub-Proyecto** (En Blanco) Anexo 4.
Salidas	▪ **Visión de Sub-Proyecto** (Diligenciado)
Roles Responsables:	Director de Sub-Proyecto.
Nombre de Actividad	Valorar/Ajustar Alcance y Riesgos del Proyecto
Descripción	Consiste en la estimación del riesgo que implica el sub-proyecto y el ajuste a los alcances y métricas del mismo, con base en la misma valoración hecha para el macro-proyecto. Esta visión debe refinarse hasta ser aprobada plenamente por el equipo (i+d) que asumió la responsabilidad de ejecutar el sub-proyecto y en colaboración con el equipo promotor (i+d) o la comunidad de desarrollo.
Entradas	▪ **Alcances del Macro-Proyecto** (Diligenciado) ▪ **Modelo Métricas del Macro-Proyecto**: **Producto y Proceso** (Específico) ▪ **Plan Preliminar de Riesgos del Macro-Proyecto** (Diligenciado) ********** ▪ **Alcances del Sub-Proyecto** (En Blanco) ▪ **Modelo Métricas del Sub-Proyecto: Producto y Proceso** (En Blanco) ▪ **Plan Preliminar de Riesgos del Sub-Proyecto** (En Blanco)
Salidas	▪ **Alcances del Sub-Proyecto** (Diligenciado) ▪ **Modelo Métricas del Sub-Proyecto: Producto y Proceso** (Específico) ▪ **Plan Preliminar de Riesgos del Sub-Proyecto** (Diligenciado)
Roles Responsables:	▪ Director de Sub-Proyecto ▪ **Colaboran:** o Comunidad de Desarrollo. o Equipo (i+d) promotor del Macro-Proyecto. o Miembros del Equipo (i+d) responsable del Sub-Proyecto.
Nombre de Actividad	Elaborar/Refinar Plan Preliminar de Proyecto
Descripción	Consiste en la elaboración preliminar de un Plan para el sub-proyecto que será ejecutado por el equipo (i+d). La información de este documento se puede encontrar en el Anexo 11. El documento diligenciado, debe ser publicado en la plataforma de integración en un área pública con el propósito de que esté disponible para consulta o referencia para toda la comunidad de desarrollo.
Entradas	▪ **Plan de Sub-Proyecto** (En Blanco) Anexo 11
Salidas	▪ **Plan de Sub-Proyecto** (Diligenciado)
Roles Responsables:	▪ Director de Sub-Proyecto ▪ **Colaboran con la revisión:** o Equipo (i+d) del Promotor del Proyecto o Comunidad de Desarrollo
Nombre de Actividad	Planificar Siguiente Iteración
Descripción	Su propósito consiste en la elaboración de un plan (de grano fino) que guiará la próxima iteración. Si existen cambios en el esfuerzo, tiempo y/ costos,

	puede ejecutarse la actividad 7.
Entradas	▪ **Plan de Iteración** (En Blanco) Anexo 5
Salidas	▪ **Plan de Iteración** (Diligenciado)
Roles Responsables:	▪ Director de Sub-Proyecto ▪ **Colaboran:** o Equipo (i+d) del Sub-Proyecto
Nombre de Actividad	Evaluar Alcances y Riesgos del Proyecto
Descripción	Luego de cada iteración, se evalúa y actualiza el estado de avance del proyecto.
Entradas	▪ **Alcances del Sub-Proyecto** (Diligenciado) ▪ **Modelo Métricas del Sub-Proyecto: Producto y Proceso** (Específico) ▪ **Plan de Riesgos del Sub-Proyecto** (Diligenciado)
Salidas	▪ **Alcances del Sub-Proyecto** (Ajustado) ▪ **Modelo Métricas del Sub-Proyecto: Producto y Proceso** (Ajustado) ▪ **Plan de Riesgos del Sub-Proyecto** (Ajustado)
Roles Responsables	▪ Director de Sub-Proyecto
Nombre de Actividad	Refinar Plan de Proyecto
Descripción	Opcionalmente, se puede actualizar el plan de proyecto (especificado en el Anexo 11, adaptando el cronograma, presupuesto y el esfuerzo con el propósito de garantizar estimaciones más fiables para las próximas iteraciones.
Entradas	▪ **Plan de Sub-Proyecto** (Diligenciado)
Salidas	▪ **Plan de Sub-Proyecto** (Ajustado)
Roles Responsables	▪ Director de Sub-Proyecto
Nombre de Actividad	Cerrar Fase
Descripción	Consiste en las actividades relacionadas con el cierre de la Fase (Inicio, Elaboración, Construcción y/o Transición). Como recomienda Larman (2003), si se dispone de una herramienta de control de versiones, usualmente se crea un punto de control etiquetado y congelado de todas las carpetas de la documentación. Ver numeral 5.4.11. A la hora de decidir si se cambia o no de fase, tenga en cuenta los hitos: Inicio (numeral 5.7.2.1), Elaboración (numeral 5.7.2.2), Construcción (numeral 5.7.2.3) y transición (numeral 5.7.2.4)
Entradas	▪ **Ninguna.**
Salidas	▪ **Ninguna.**
Roles Responsables	▪ Director de Sub-Proyecto
Nombre de Actividad	Cerrar Proyecto
Descripción	Consiste en las actividades relacionadas con el cierre del Proyecto. Como recomienda Larman (2003), si se dispone de una herramienta de control de versiones, usualmente se crea un punto de control etiquetado y congelado de todas las carpetas de la documentación. Ver numeral 5.4.11.
Entradas	▪ **Ninguna.**
Salidas	▪ **Ninguna.**
Roles Responsables	▪ Director de Sub-Proyecto
Nombre de Actividad	Final de Iteración
Descripción	Consiste en las actividades relacionadas con el cierre de la iteración actual. Como recomienda Larman (2003), si se dispone de una herramienta de control de versiones, usualmente se crea un punto de control etiquetado y congelado de todas las carpetas de la documentación. Ver numeral 5.4.11.

Entradas	▪ **Ninguna.**
Salidas	▪ **Ninguna.**
Roles Responsables	▪ Director de Sub-Proyecto

Tabla 20: Actividades del Flujo de Gestión a nivel de Sub-Proyecto

5.11.5. Artefactos

Los artefactos elaborados, refinados o involucrados durante la ejecución de las actividades de esta disciplina se relacionan en la siguiente tabla:

No.	Nombre	Propósito	Especificado en…
1	Visión del Macro-Proyecto	Provee una breve descripción acerca del problema y objetivos perseguidos.	Anexo 4
2	Alcances del Macro Proyecto	Provee una breve descripción de los resultados esperados y sus correspondientes indicadores de logro.	Anexo 11
3	Modelo de Métricas del Macro-Proyecto: Producto y Proceso	Especifica un sistema de métricas para el producto y proceso de software con el propósito de valorar la calidad.	Anexo 1 y Anexo 9
4	Plan Preliminar de Riesgos del Macro-Proyecto	Tabula una colección preliminar de riesgos así como su correspondiente descripción y estrategia de mitigación.	Anexo 6
5	Distribución Macro-Proyecto	Propone una descomposición del macro-proyecto en sub-proyectos mas pequeños que puedan ser asignados a equipos (i+d) de la comunidad de desarrollo.	Anexo 11
6	Valoración: Gestión Riesgos, Monitoreo y Control	Consiste en una determinación del nivel de mitigación de los riesgos del proyecto así como su nivel de avance.	Anexo 10
7	Visión del Sub-Proyecto	Es similar a la visión del macro-proyecto pero con el contexto del sub-problema y objetivos limitado al sub-proyecto.	Anexo 4
8	Alcances del Sub-Proyecto	Similar a los alcances del macro-proyecto pero con el contexto concreto de los productos e indicadores del sub-proyecto.	Anexo 11
9	Modelo Métricas del Sub-Proyecto: Producto y Proceso	Especifica un sub-conjunto de métricas del producto y proceso a partir del modelo general.	Anexo 1 y Anexo 9
10	Plan Preliminar de Riesgos del Sub-Proyecto	Consiste en un sub-conjunto de riesgos seleccionado a partir de la lista preliminar de riesgos para el	Anexo 6

		macro-proyecto.	
11	Plan de Sub-Proyecto	Consiste en una especificación de las dimensiones de producto, personas, proceso, tiempo y presupuesto en el contexto del sub-proyecto específico.	Anexo 11
12	Plan de Riesgos del Sub-Proyecto	Consiste en una especificación de las estrategias correspondientes a cada uno de los riesgos detectados con el propósito de mitigarlos.	Anexo 10

Tabla 21: Artefactos de la Disciplina Gestión del Proyecto

5.11.6. Arquitectura

A continuación, se proporcionan una breve descripción relativa a los diagramas de clase del análisis en UML para los sub-modelos: “Planificación” y “Monitoreo-Control, Gestión de Riesgos”.

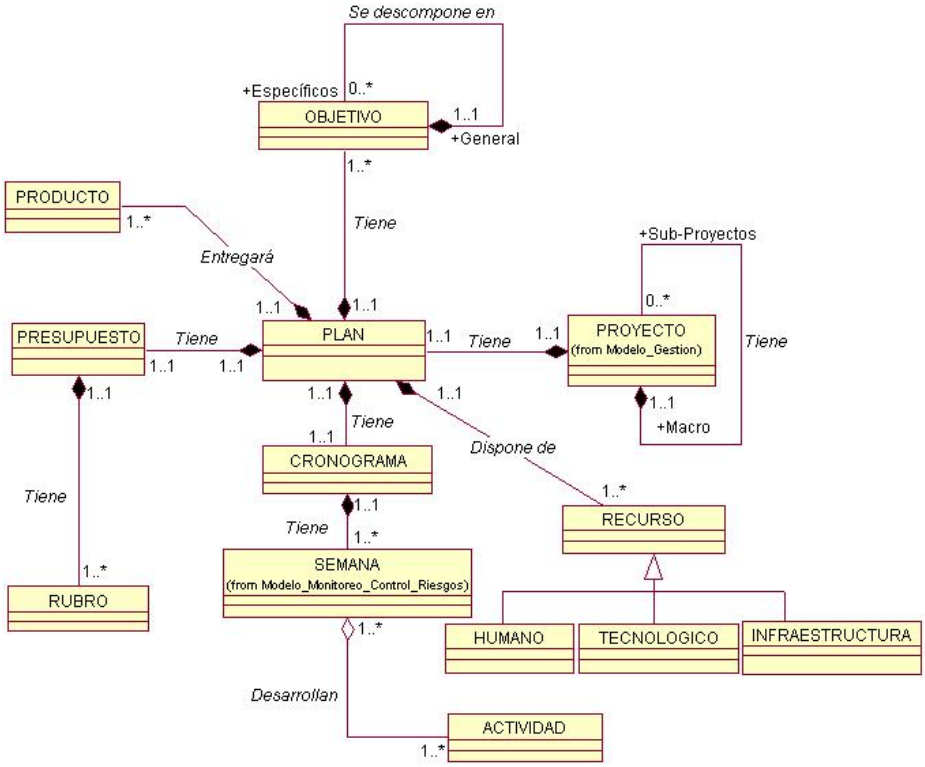

Figura 44: Arquitectura del Sub-Modelo de Planificación

En el diagrama de la Figura 44: Arquitectura del Sub-Modelo de Planificación se ilustran los componentes fundamentales de esta disciplina. Como punto central se encuentra el proyecto, el cual puede descomponerse a su vez en sub-proyectos. Cada proyecto contempla un plan que especifica las dimensiones del producto, el cronograma, los objetivos, presupuesto y los recursos (humanos, tecnología e infraestructura). La planificación temporal del cronograma se divide por semanas durante las cuales, se ejecutan las actividades del proyecto. En el diagrama de la Figura 45: Arquitectura del Sub-Modelo "Monitoreo, Control y Riesgos", se ilustran los elementos que garantizan el monitoreo y control, mediante las métricas (las cuáles pueden compartirse para varios proyectos) y sus metas asociadas. De otra parte, los elementos que implementan la gestión de riesgos, estarían representados en primer lugar por los riesgos y sus correspondientes métodos de control, y en segundo lugar, por la mitigación de los riesgos, reflejada en los elementos de progreso a través de las semanas. El artefacto para la especificación de los requisitos funcionales y no-funcionales del producto software, se encuentra descrito en detalle en el Anexo 12.

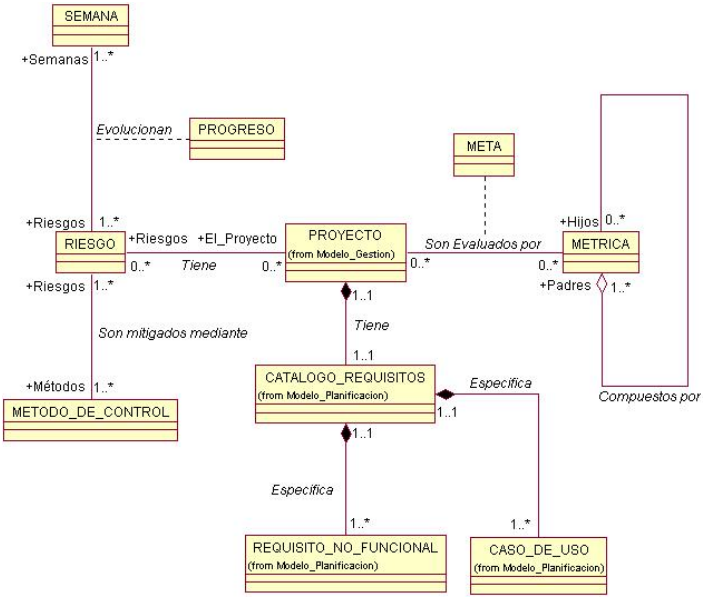

Figura 45: Arquitectura del Sub-Modelo "Monitoreo, Control y Riesgos"

5.11.7. Premisas

Las siguientes premisas se consideran verdaderas para la ejecución del modelo de gestión propuesto por esta metodología:

- Existe una plataforma de integración sobre Internet, la cuál soporta los procedimientos relacionados con la gestión de macro-proyectos y sub-proyectos.

- Cada gerente de proyecto puede obtener una visión global de otros proyectos, sin derecho de realizar alteraciones en su información a menos que también sea gerente.

- Los requisitos de cada proyecto no pueden ser alterados por otro equipo de desarrollo, pero si pueden debatirse y negociarse, mediante los servicios de foro de la plataforma de integración.

- La plataforma de integración debe ser accesible por todos y cada uno de los miembros de la comunidad, desde cualquier ubicación geográfica, salvo casos de fuerza mayor.

- Debe existir una configuración de equipo con programador jefe (explicado en el numeral 5.8.1.2) disponible para ejecutar las responsabilidades de esta disciplina.

- Es fundamental que exista un modelo de documentación que permita instrumentar esta disciplina. Se recomienda la estructura de directorios del Anexo 5. Allí se muestra la ubicación exacta de los artefactos de esta disciplina en el marco de una iteración para cualquier fase del proceso. La idea es que con cada nueva iteración, el usuario de Agile-DISOP, sólo

replicará el sub-directorio Iteración[66], logrando con esto, alcanzar una estructura de documentación uniforme para todos los proyectos que emprenda la comunidad.

5.11.8. Métricas

Las métricas en este contexto, ayudan a valorar la calidad de la planificación respecto de proyectos ya finalizados. A través de valores cuantitativos, se pueden comparar los costos y tiempos asociados a proyectos de tamaño semejante, con el propósito de realizar cada vez, una planificación mas refinada y exacta. Para tal efecto, se hace necesario utilizar las siguientes métricas de proyecto:

- Métricas para la Estimación del Tamaño, Costo, Esfuerzo y Tiempo del Proyecto.

- Histórico de los proyectos ejecutados con anterioridad con el propósito de asignar recursos y tiempo a las actividades de proyectos de tamaño y complejidad semejante. Así mismo, realizar una asignación de responsabilidades al personal sobre una base de experiencia pasada.

- Valorar la calidad de los artefactos elaborados dentro de la Gestión de proyectos y en especial la “Visión del Proyecto” (Anexo 4) y “Plan de Proyecto” (Anexo 11).

- **Valor Ganado.** Esta métrica es muy importante para el seguimiento de la cantidad de trabajo realizado. Esta métrica se calcula de la siguiente forma:

[66] Asignándole el número de iteración al directorio recién creado.

- $VG = Tamaño * \left[\sum_{i=1}^{p} Ci \right] * \left[\frac{\sum_{i=1}^{p} CiPi}{\sum_{i=1}^{p} Ci} \right]$

- **Donde:**

  - Tamaño: Se mide en líneas de código.

  - $Pi = \frac{Unidades\_Software\_Por\_Actividad\_i}{total\_número\_de\_Unidades\_\text{Re}queridas\_Actividad\_i}$

  - $Ci = \frac{Horas\_de\_\text{Pr}ogramación\_Actividad\_i}{Tamaño\_Lineas\_Código\_Unidad\_i}$

5.11.9. Métodos

Se recomienda la utilización de los diagramas de Gantt y PERT con el propósito de obtener:

- Fecha más temprana para el inicio y finalización de una actividad. Fecha más tardía para el inicio y finalización para una actividad. Camino crítico de actividades para el proyecto.

- Especificar dependencias entre actividades, incluso entre proyectos correlacionados. Calcular el tiempo total del proyecto.

- Se recomienda utilizar técnicas gramaticales para la elaboración correcta de los artefactos de solicitud y plan de proyecto, estos artefactos están auto-documentados con indicaciones de cómo deben redactarse, sin embargo.

en el Anexo 14[67] se compilan algunas recomendaciones adicionales generales del arte de redactar.

5.12. DISCIPLINA: PROCESO DE DESARROLO DE AGILE-DISOP

5.12.1. Propósito

El Propósito de esta disciplina consiste en proveer el marco de referencia para la ejecución de las sub-disciplinas de la ingeniería del software como son: Requerimientos, Análisis-Diseño, Implementación y Pruebas. Esta disciplina está centrada exclusivamente en el desarrollo del producto de software.

5.12.2. Conceptos

Los conceptos involucrados en esta disciplina son: Análisis, Diseño, Implementación, Pruebas. Para encontrar las definiciones correspondientes a los anteriores conceptos favor referirse al Anexo 3. A continuación en el numeral 5.12.3 se describe el flujo de trabajo para esta disciplina.

5.12.3. Flujo de Trabajo

En la Figura 46, se ilustra la secuencia de actividades que se siguen durante cualquier iteración de **Agile-DISOP,** en el marco del desarrollo de un sub-proyecto que ejecuta un sub-equipo de la comunidad. Puede observarse que el esquema propuesto es en esencia una cascada de actividades desde el análisis hasta las pruebas. Antes de iniciar esta secuencia debe estimarse en el "Plan de Iteración"[68], todo el esfuerzo, tiempo, dinero, personal y demás recursos requeridos para realizarla. Debe tenerse en cuenta que el flujo de trabajo propuesto, se ejecuta tomando sólo algunos casos de uso (estrechamente relacionados) y su duración no

---

[67] Disponible sólo en formato digital
[68] Véase Figura 43 y Tabla 20: Actividades del Flujo de Gestión a nivel de Sub-Proyecto

debería extenderse más allá de 8 semanas, de lo contrario se esfumaría el espíritu iterativo-incremental del proceso y se tornaría en un proceso en cascada permanente. Otra cosa, las líneas punteadas establecen una relación de dependencia entre actividades, en lugar de una relación de secuencia. Por último, debe recordarse que este flujo de trabajo, tiene como propósito fundamental, la publicación de un incremento software[69] operacional ante la comunidad de desarrollo.

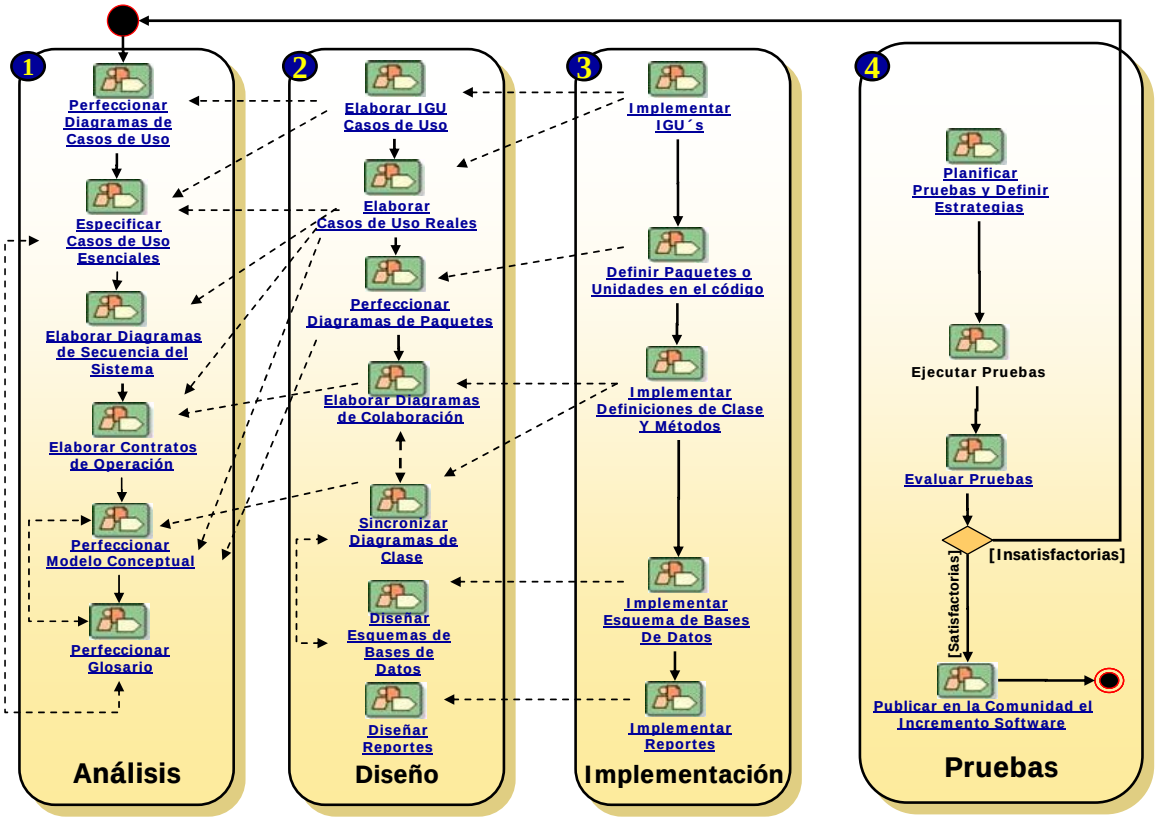

Figura 46: Flujo de Trabajo para la Disciplina de Desarrollo

[69] El cuál realiza y/o implementa el/los caso(s) de uso(s) tomados para la iteración de turno.

5.12.4. Actividades

Nombre de Actividad	**Análisis::** Perfeccionar Diagramas de Casos de Uso
Descripción	El diagrama de Casos de Uso describe gráficamente la interacción que ocurre entre los actores y el sistema, sin embargo debe irse refinando a medida que se avanza con el desarrollo del proyecto. Es posible que se encuentren nuevos casos de uso, se re-organicen los ya existentes o se establezcan nuevas relaciones con otros actores o nuevas dependencias con otros casos de uso.
Entradas	▪ **Diagramas de Casos de Uso (Versión anterior) Véase Anexo 5**
Salidas	▪ **Diagramas de Casos de Uso (Nueva Versión)**.
Roles Responsables:	▪ Analista (RUP 2003)
Nombre de Actividad	**Análisis::** Especificar Casos de Uso Esenciales
Descripción	El caso de uso, es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso. Los casos de uso son historias o casos de utilización de un sistema; ejemplifican e incluyen tácitamente los requerimientos en las historias que narran. Existen dos momentos para especificar un caso de uso esencial: • **Un caso de uso de alto nivel** se utiliza para lograr rápidamente un entendimiento de los principales procesos globales. Incluye una descripción breve sobre el inicio, desarrollo y finalización del caso de uso. • **Un caso de uso en formato expandido** muestra más detalles que el anterior y se utiliza a menudo para entender y conocer más profundamente los procesos y los requerimientos. Está escrito en términos de los estímulos enviados por el actor o actores y las reacciones o respuestas del sistema a esos estímulos.
Entradas	▪ Diagramas de Casos de Uso. **Ver Anexo 5**
Salidas	▪ **Especificación Extendida para los Casos de Uso. Véase Anexo 5**
Roles Responsables:	▪ Especificador de Requerimientos (RUP 2003)
Nombre de Actividad	**Análisis::** Elaborar Diagramas de Secuencia del Sistema
Descripción	Los casos de uso indican cómo los actores interactúan con el sistema de software que es en realidad lo que se desea desarrollar. Durante la interacción, un actor genera eventos dirigidos al sistema, solicitando de este, la ejecución de una operación a cambio. El diagrama de secuencia del sistema, es una representación gráfica que muestra, en determinado escenario de un caso de uso, los eventos generados por actores externos, su orden y los eventos internos del sistema. En un diagrama de secuencia del sistema, este último, se trata como una caja negra; los diagramas se centran en los eventos que fluyen desde los actores externos hacia el sistema.
Entradas	▪ **Especificación Extendida para Casos de Uso**. **Véase Anexo 5**
Salidas	▪ Diagrama de Secuencia del Sistema para los Casos de Uso. **Ver Anexo 5**
Roles Responsables:	▪ Diseñador RUP (2003)
Nombre de Actividad	**Análisis::** Elaborar Contratos de Operación
Descripción	Un contrato es un documento que describe lo que una operación se propone lograr. Suele elaborarse en sentido declarativo, enfatizando en lo que sucederá y no como se conseguirá. Los contratos suelen expresarse a partir

	de los cambios ocurridos entre pre-condiciones y pos-condiciones. El contrato de operación describe los cambios de estado del sistema total cuando se llama o ejecuta una de sus operaciones. Para preparar contratos en los casos de uso deben tenerse en cuenta las siguientes directrices: ▪ Identifique las operaciones del sistema mediante los diagramas de secuencia. ▪ Elabore un contrato para cada operación identificada. ▪ En la plantilla del contrato, comience redactando primero la sección de Responsabilidades describiendo informalmente el propósito de la operación en esta sección. ▪ A continuación redacte la sección Poscondiciones describiendo en forma declarativa uno a uno los cambios de estado en los objetos involucrados en la realización de la operación. Estos objetos deben existir en el modelo conceptual, en caso contrario debe examinarse su inclusión a este. Utilice las siguientes categorías para describir poscondiciones: • Creación y Eliminación de Instancias. • Modificación de Atributos. • Asociaciones formadas y canceladas.
Entradas	▪ Casos de Uso Expandidos y Esenciales. **Véase Anexo 5** ▪ Diagramas de Secuencia del Sistema. **Véase Anexo 5**
Salidas	▪ **Contratos de Operación para cada Caso de Uso.**
Roles Responsables:	▪ Especificador de Requerimientos. (RUP 2003)
Nombre de Actividad	**Análisis::** Perfeccionar el Modelo Conceptual
Descripción	El modelo conceptual explica los conceptos significativos en un dominio del problema; es el artefacto más importante a crear durante el análisis orientado a objetos. Identificar muchos objetos o conceptos constituye la esencia del análisis orientado a objetos. Un modelo conceptual representa cosas del mundo real, no componentes de software y muestra: Conceptos, Asociaciones entre Conceptos y Atributos de Conceptos. Para identificar conceptos puede utilizarse la siguiente lista de categorías: Objetos físicos o tangibles, Especificaciones, Diseños o Descripciones de Cosas, Lugares, Transacciones, Línea o renglón de elemento en una transacción, Papel de las Personas, Contenedores de otras cosas, Cosas dentro de un contenedor, Otros sistemas de computo o electromecánicos externos al sistema, Conceptos de nombres abstractos, Organizaciones, Eventos, Procesos, Reglas y Políticas, Catálogos, Registros de Finanzas, de Contratos, de Asuntos legales, Instrumentos y Servicios Financieros, Manuales y Libros.
Entradas	▪ Diagramas de Casos de Uso y Casos de Uso Esenciales. **Ver Anexo 5** ▪ Diagramas de Secuencia del Sistema y Contratos de Operación. **Anexo 5** ▪ Glosario. **Ver Anexo 5**
Salidas	▪ **Diagrama de Clases del Análisis o Modelo Conceptual. Ver Anexo 5**
Roles Responsables	▪ Diseñador. (RUP 2003)
Nombre de Actividad	**Análisis::** Perfeccionar el Glosario
Descripción	Un glosario es un documento simple en el cuál se definen términos. Este incluye y define todos los términos que requieren explicación para mejorar la comunicación y aminorar el riesgo de malos entendidos. Un significado uniforme y compartido resulta extremadamente importante durante el desarrollo de las aplicaciones, sobre todo cuando muchos miembros del equipo intervienen en el proyecto.
Entradas	▪ Diagramas de Casos de Uso y Casos de Uso Esenciales.

	▪ Diagramas de Secuencia del Sistema y Contratos de Operación. ▪ Modelo Conceptual
Salidas	▪ **Diagrama de Clases del Análisis o Modelo Conceptual. Ver Anexo 5**
Roles Responsables	▪ Especificador de Requisitos. (RUP 2003)

Tabla 22: Actividades del Flujo de Desarrollo durante el Análisis

Nombre de Actividad	**Diseño::** Elaborar IGU para Casos de Uso
Descripción	La Interfaz gráfica de usuario es el mecanismo de interacción que dispone el sistema en el marco de un caso de uso para aceptar eventos del actor. En esta actividad se elaboran los diseños preliminares de la (s) interfaz (ces) gráfica (s) de usuario para el (los) caso (s) de uso tomados para esta iteración.
Entradas	▪ Diagramas de Casos de Uso y Casos de Uso Esenciales. **Ver Anexo 5**
Salidas	▪ **Interfaz Gráfica de Usuario – IGU para Casos de Uso. Ver Anexo 5**
Roles Responsables	▪ Diseñador de Interfaz Gráfica (RUP 2003)
Nombre de Actividad	**Diseño::** Elaborar Casos de Uso Reales.
Descripción	La versión "Real" del caso de uso, es simplemente un refinamiento de la versión extendida. En esta versión, la interacción entre el actor y el sistema se desarrolla en el marco de una interfaz gráfica de usuario. Los estímulos del actor sobre el sistema, ahora se convierten en acciones directas (clicks, arrastre, selección y chequeo) sobre botones, cuadros de texto, combos de selección, etc.
Entradas	▪ Diagramas de Casos de Uso y Casos de Uso Esenciales. ▪ Interfaz Gráfica de Usuario para los Casos de Uso. ▪ Diagramas de Secuencia del Sistema y Contratos de Operación. ▪ Modelo Conceptual. **Ver Anexo 5**
Salidas	▪ **Caso de Uso Real. Ver Anexo 5**
Roles Responsables	▪ Especificador de Requisitos. (RUP 2003) ▪ Diseñador de Interfaces Gráficas. (RUP 2003)
Nombre de Actividad	**Diseño::** Perfeccionar Diagrama de Paquetes
Descripción	Los diagramas de paquetes ayudan a reducir la complejidad de la arquitectura por medio del empaquetamiento de clases en unidades lógicas (paquetes) aisladas, pero dependientes. Por medio del empaquetamiento, se puede organizar la arquitectura en vistas más pequeñas y comprensibles aumentando su "Entendibilidad".
Entradas	▪ Diagrama de Clases. **Ver Anexo 5**
Salidas	▪ **Diagrama de Paquetes. Ver Anexo 5**
Roles Responsables	▪ Diseñador (RUP 2003)
Nombre de Actividad	**Diseño::** Elaborar Diagramas de Colaboración.
Descripción	Los diagramas de colaboración ilustran la secuencia de mensajes que intercambian los objetos en tiempo de ejecución del sistema para satisfacer las operaciones descritas en los contratos de operación elaborados para el caso de uso. Cada diagrama de colaboración puede contener mensajes relacionados con: ▪ Creación / Destrucción de Objetos, Asociación / Disociación de Objetos ▪ Iteración, Decisión, Búsqueda / Recuperación de Objetos y Asignación. Estos artefactos son esenciales a la hora de determinar el esqueleto del código de todos y cada uno de los métodos que están representados como

	mensajes en cada diagrama. Además puede usarse también para refinar la interfaz de cada clase, al asignarle métodos.
Entradas	▪ Diagrama de Clases. **Ver Anexo 5**
Salidas	▪ **Diagramas de Colaboración. Ver Anexo 5**
Roles Responsables	▪ Diseñador. (RUP 2003)
Nombre de Actividad	**Diseño::** Sincronizar los Diagramas de Clase
Descripción	Una vez realizados los diagramas de colaboración (uno por cada operación del caso de uso de turno), se deben sincronizar los diagramas de clases del diseño, creando las definiciones de operaciones en las clases involucradas en el diagrama de colaboración, a partir de los mensajes presentes en el diagrama. Un método se define para una clase, siempre que un objeto o instancia de esta, recibe un mensaje desde otro objeto o instancia presente en el diagrama de colaboración. Por último, si ya existe un intento anterior de implementar un esquema de bases de datos, es posible que también el diagrama de clases se refine a partir de aquel.
Entradas	▪ Diagramas de Colaboración. **Ver Anexo 5** ▪ Esquemas de Bases de Datos. **Ver Anexo 5**
Salidas	▪ **Diagrama de Clases. Ver Anexo 5**
Roles Responsables	▪ Diseñador (RUP 2003)
Nombre de Actividad	**Diseño::** Diseñar Esquemas de Bases de Datos
Descripción	En caso de que el sistema utilice bases de datos, deben crearse las tablas correspondientes a las clases que intervienen en los casos de uso tomados para esta iteración.
Entradas	▪ Diagramas de Clases **Ver Anexo 5**
Salidas	▪ **Diseño del Esquema de Bases de Datos. Ver Anexo 5**
Roles Responsables	▪ Diseñador de la Base de Datos. (RUP 2003)
Nombre de Actividad	**Diseño::** Diseñar Reportes
Descripción	A partir de los requisitos de información del sistema se pueden inferir los reportes deseados para este último. Cada reporte debe tener un propósito bien definido. Su estructura se divide en tres grandes partes: Cabecera, Detalle y Pié. Pueden incluir gráficos, tablas, formulas o condiciones lógicas. Por último es muy importante que cada reporte se pueda imprimir con una alta fidelidad.
Entradas	▪ Diseño del Esquema de Bases de Datos **Ver Anexo 5**
Salidas	▪ **Diagrama de Paquetes. Ver Anexo 5**
Roles Responsables	▪ Diseñador de la Base de Datos (RUP 2003)

Tabla 23: Actividades del Flujo de Desarrollo Durante el Diseño

Nombre de Actividad	**Implementación::** Implementar IGU´s
Descripción	Consiste en la creación de formularios, ventanas o cuadros de diálogo de cada caso de uso involucrado en la presente iteración, teniendo muy en cuenta las IGU´s y los casos de uso reales elaborados durante el diseño. Esta actividad se realiza en el propio entorno de desarrollo.
Entradas	▪ IGU´s Diseñadas **Ver Anexo 5** ▪ Casos de Uso Reales **Ver Anexo 5**

Salidas	▪ **IGU´s Implementadas Ver Anexo 5**
Roles Responsables	▪ Diseñador de Interfaces Gráficas (RUP 2003) ▪ Desarrollador (RUP 2003)
Nombre de Actividad	**Implementación::**Definir Paquetes o Unidades en el Código
Descripción	Esta actividad tiene como propósito fundamental, reflejar los diagramas de paquetes del diseño en el código fuente. Usualmente y según el lenguaje, cada paquete gráfico se corresponderá con un **Package**, **Unit** o **NameSpace** en el código fuente. Es vital que el código se corresponda fielmente con los diagramas de paquetes para guardar los atributos de "correctitud", "completitud" y "entendibilidad" de la arquitectura software.
Entradas	▪ Diagrama de Paquetes del Diseño
Salidas	▪ **Código Fuente con Paquetes Definidos.**
Roles Responsables	▪ Diseñador. (RUP 2003) ▪ Desarrollador. (RUP 2003)
Nombre de Actividad	**Implementación::** Implementar Definiciones de Clase y métodos.
Descripción	Esta actividad tiene como propósito fundamental, reflejar los diagramas de clases del diseño en el código fuente. Usualmente y según el lenguaje, cada clase en el diagrama se corresponderá con un **Class** en el código fuente. Es vital que el código se corresponda fielmente con los diagramas de clases para guardar los atributos de "correctitud", "completitud" y "entendibilidad" de la arquitectura software. Por otra parte, los diagramas de colaboración juegan un papel vital a la hora de definir la implementación de los métodos de una clase, en virtud de que facilitan enormemente la concepción del código fuente que pertenece a un método determinado.
Entradas	▪ Diagramas de Colaboración. **Ver Anexo 5** ▪ Diagramas de Clase. **Ver Anexo 5**
Salidas	▪ **Clases de la Implementación. Ver Anexo 5**
Roles Responsables	▪ Diseñador ▪ Desarrollador
Nombre de Actividad	**Implementación::**Implementar Esquemas de Bases de Datos
Descripción	En caso de que el sistema utilice un mecanismo de almacenamiento persistente (Ej. Una Base de Datos), esta actividad se concentra en el mapeo del esquema de bases de datos diseñado para la actual iteración, a un esquema implementado en un motor de bases de datos elegido.
Entradas	▪ Diseño del Esquema de Bases de Datos. **Ver Anexo 5**
Salidas	▪ **Esquema de Bases de Datos Implementado. Ver Anexo 5**
Roles Responsables	▪ Diseñador de Bases de Datos. (RUP 2003)
Nombre de Actividad	**Implementación::**Implementar Reportes
Descripción	En caso de que el sistema utilice un mecanismo de almacenamiento persistente (Ej. Una Base de Datos), esta actividad se concentra en la implementación de los reportes diseñados para la actual iteración, sobre el motor de bases de datos elegido.
Entradas	▪ Diseño de Reportes. **Ver Anexo 5**
Salidas	▪ **Reportes Implementados. Ver Anexo 5**
Roles Responsables	▪ Diseñador de Bases de Datos. (RUP 2003)

Tabla 24: Actividades del Flujo de Desarrollo Durante la Implementación

Nombre de Actividad	**Pruebas::** Planificar Pruebas y Definir Estrategias
Descripción	Esta actividad tiene como propósito fundamental, la identificación de información y componentes de software del proyecto que se eligen durante esta iteración para las pruebas, la enumeración de los requerimientos para realizar las pruebas, recomendar y describir las estrategias de pruebas que se emplearan, identificar los recursos y esfuerzo requeridos para realizar las pruebas y finalmente enumerar los productos entregables al final de las pruebas. Por otra parte, la estrategia de pruebas define: • Las herramientas y técnicas que se usarán. • Los criterios de finalización exitosa de las pruebas.
Entradas	▪ Plan de Proyecto. ▪ Plan de Iteración. ▪ Modelo del Análisis – Modelo del Diseño – Modelo de la Implementación.
Salidas	▪ **Plan de Pruebas de la Iteración. Ver Anexo 5**
Roles Responsables	▪ Gestor del Proyecto (RUP 2003)
Nombre de Actividad	**Pruebas::**Ejecutar Pruebas
Descripción	Consiste en implementar las pruebas sobre los componentes de software elegidos para esta iteración, aplicando las estrategias concebidas.
Entradas	▪ Modelo del Análisis – Modelo del Diseño – Modelo de la Implementación. ▪ Plan de Pruebas de la Iteración.
Salidas	▪ **Registro de los Resultados de las Pruebas. Ver Prototipo Anexo 16ª**
Roles Responsables	▪ Ingeniero de Pruebas
Nombre de Actividad	**Pruebas::** Evaluar Pruebas
Descripción	Consiste en la consolidación, síntesis e interpretación de los resultados de las pruebas con el objeto de fundamentar la toma de decisiones relacionadas con el rediseño o no de la arquitectura. De la satisfacción de estos resultados depende que el incremento de software sea publicado o no en la plataforma.
Entradas	▪ Registro de los Resultados de las Pruebas.
Salidas	▪ **Interpretación de Resultados. Ver Prototipo Anexo 16b**
Roles Responsables	▪ Analista de Pruebas
Nombre de Actividad	**Pruebas::**Publicar en la Comunidad el Incremento de Software
Descripción	Con base en la interpretación de resultados de las pruebas, se valora la estabilidad y calidad del producto o incremento software desarrollado para la actual iteración. Si el producto satisface los requerimientos funcionales y no funcionales tomados durante la iteración, entonces se publica a toda la comunidad para ser integrado como un componente software estable que hace parte del producto final. En caso contrario, se inicia nuevamente desde la sub-disciplina del análisis-diseño-implementación para corregir los defectos descubiertos durante las pruebas.
Entradas	▪ Interpretación de Resultados.
Salidas	▪ **Incremento Software**
Roles Responsables	▪ Equipo (i+d)

Tabla 25: Actividades del Flujo de Desarrollo Durante las Pruebas

5.12.5. Artefactos

Los artefactos elaborados, refinados o involucrados durante la ejecución de las actividades de esta disciplina se relacionan en la siguiente tabla:

No.	Nombre	Propósito	Especificado en…
1	Diagrama de Casos de Uso	Mostrar los casos de uso del sistema y la relación con sus actores.	Anexo 5
2	Caso de Uso (Alto Nivel)	Especifica brevemente la historia de interacción entre los actores y el caso de uso.	Anexo 5
3	Caso de Uso (Expandido)	Extiende la declaración del caso de uso de alto nivel, incluyendo escenarios y excepciones.	Anexo 5
3	IGU	Representa la interfaz gráfica de usuario para la interacción en el marco de uno o varios casos de uso.	Anexo 5
4	Caso de Uso (Real)	Refina la especificación del caso de uso expandido integrando la IGU diseñada.	Anexo 5
5	Diagrama de Clases del Análisis	Muestra conceptos del dominio, sus atributos y asociaciones.	Anexo 5
6	Diagrama de Secuencia	Muestra como se implementa una operación, mediante el intercambio de mensajes de los objetos en tiempo de ejecución.	Anexo 5
7	Contratos de Operaciones	Define cual es el estado de información del sistema, antes y después de que se ejecute una operación.	Anexo 5
8	Diagrama de Clases del Diseño	Muestra las clases de un sistema o sub-sistema, incluyendo los atributos, métodos y asociaciones de cada clase.	Anexo 5
9	Diagrama de Colaboración	Igual que el Diag. De Secuencia.	Anexo 5
10	Diagrama de Paquetes	Organiza visualmente la arquitectura mediante el uso de contenedores lógicos llamados paquetes. Cada paquete puede contener a su vez a otros sub-paquetes o simplemente clases.	Anexo 5
11	Esquema de Bases de Datos	Es la representación de Bases de Datos Relacionales, XML, o archivos planos que le otorgan persistencia a la información del sistema.	Anexo 5
12	Reportes	Consisten en los informes que arrojan resultados a partir de la base de datos.	Anexo 5

13	Diccionario de Datos	Reúne y define todos los términos del problema o negocio.	Anexo 5
14	Plan De Pruebas de la Iteración	Determina los recursos, componentes, esfuerzo, personal y dinero que se destinan para realizar las pruebas de software durante una iteración.	Anexo 5
15	Registro Resultados de Pruebas	Organiza los datos que arrojan las pruebas hechas al software.	Anexo 5
16	Interpretación de Resultados de Pruebas	Consolida y sintetiza los datos de las pruebas mediante gráficos y tablas.	Anexo 5
17	Incremento Software	Es un componente software operacional que implementa todos los casos de uso tomados para una iteración.	Anexo 5

Tabla 26: Artefactos de la Disciplina Desarrollo

5.12.6. Arquitectura

Con el propósito de facilitar la lectura del presente documento, el lector encontrará la arquitectura de esta disciplina en el Anexo 5, específicamente en la ruta siguiente:

- Anexos\Anexo5_Modelo_Documentacion\Agile_DISOP\Anexos\Disciplina_Desarrollo.mdl.

5.12.7. Premisas

- Debe existir una configuración de equipo con programador jefe (explicado en el numeral 5.8.1.2) disponible para la resolución ágil de problemas que puedan surgir con la instalación, puesta en marcha y mantenimiento de la plataforma ante toda la comunidad de desarrollo.

- Es fundamental que exista un modelo de documentación que permita instrumentar esta disciplina. Se recomienda la estructura de directorios del Anexo 5. Allí se muestra la ubicación exacta de los artefactos de esta disciplina en el marco de una iteración para cualquier fase del proceso. La

idea es que con cada nueva iteración, el usuario de Agile-DISOP, sólo replicará el sub-directorio Iteración[70], logrando con esto, alcanzar una estructura de documentación uniforme para todos los proyectos que emprenda la comunidad.

5.12.8. Métricas

Las métricas recomendadas para empezar son las descritas en el Anexo 1 e implementadas en el Anexo 16.

5.12.9. Métodos

Se recomiendan los métodos orientados a objetos para las sub-disciplinas de: Análisis, Diseño, Implementación y Pruebas.

5.13. DISCIPLINA: MEJORAMIENTO DE AGILE-DISOP

5.13.1. Propósito

Esta disciplina tiene como objeto fundamental, la concepción y puesta en marcha de mecanismos eficaces para emprender un esfuerzo de mejoramiento contínuo sobre el mismo proceso de desarrollo de **Agile-DISOP.** Se ha determinado que el modelo de mejora IMPACT[71] (Scott 2002) es un buen punto de partida para, concebir el modelo de mejora para **Agile-DISOP**, principalmente porque presenta características clave para que pequeñas organizaciones[72] lo puedan adoptar iterativa e incrementalmente con poca inversión y esfuerzo alcanzándoles resultados tangibles en un tiempo razonable.

---

[70] Asignándole el número de iteración al directorio recién creado.
[71] Especificado en el numeral 3.3
[72] Equivalentes a los equipos (i+d) geográficamente dispersos en una comunidad de desarrollo.

5.13.2. Conceptos

Los conceptos involucrados en esta disciplina son: Área de Mejora, Estrategia de Mejora, Guía del Proceso. Para encontrar las definiciones correspondientes a los anteriores conceptos favor referirse al Anexo 3.

5.13.3. Flujo de Trabajo

Como puede observase en la Figura 47: Flujo de Trabajo para el Mejoramiento del Proceso, se consideran dos niveles para el proceso de mejoramiento del mismo **Agile-DISOP**. En primer lugar se especifica el nivel del proyecto (la parte derecha) que es donde se escogen las áreas que son objetivo del mejoramiento, luego las estrategias implementadas son valoradas por su impacto en el desempeño del proyecto. En el nivel del proceso, las estrategias exitosas son incorporadas y propagadas a nuevos proyectos que emprenda la comunidad.

(Espacio en blanco intencional para mantener el formato del documento)

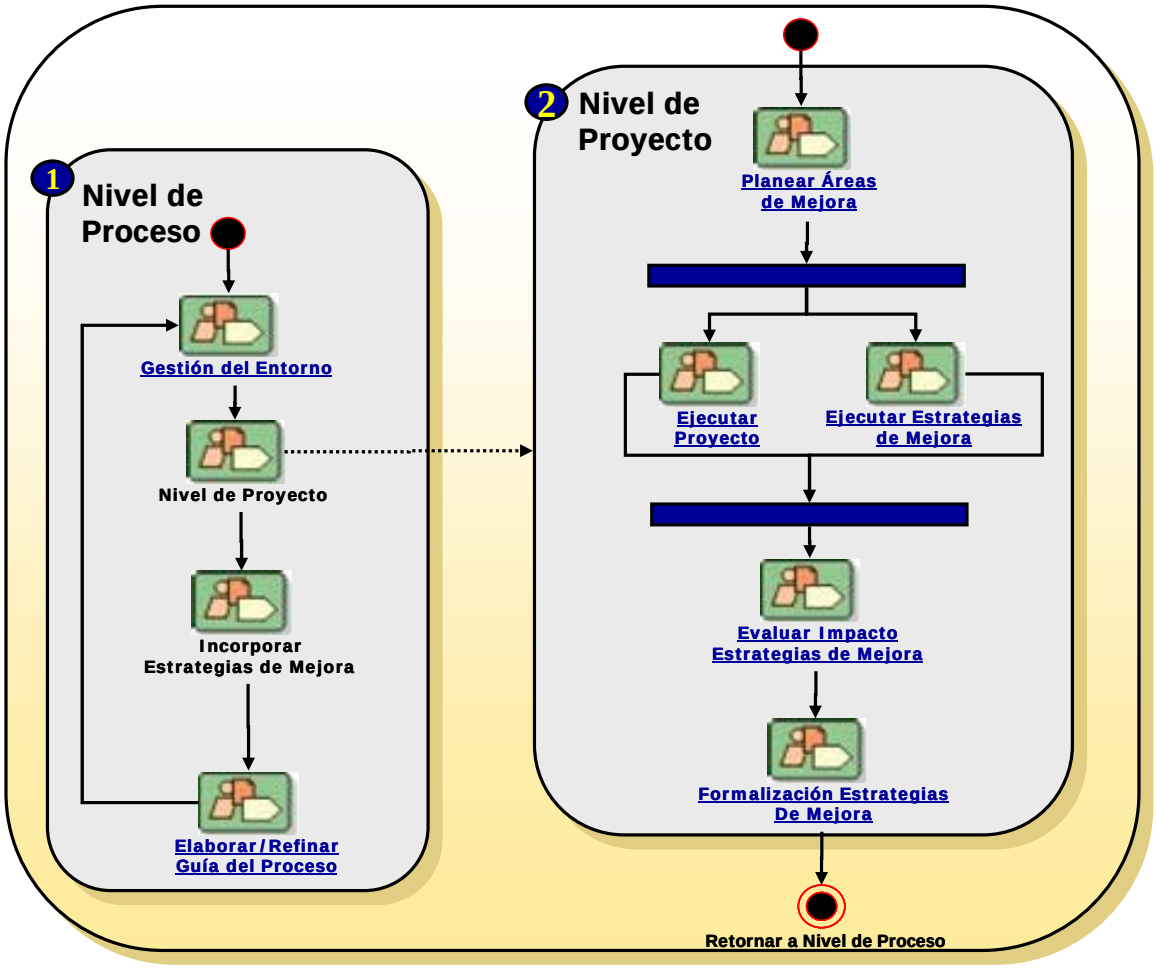

Figura 47: Flujo de Trabajo para el Mejoramiento del Proceso

5.13.4. Actividades

La descripción de las actividades de la Figura 47: Flujo de Trabajo para el Mejoramiento del Proceso se especifica a continuación:

NIVEL DEL PROCESO	
Nombre de Actividad	Gestión del Entorno
Descripción	Su especificación corresponde al flujo de trabajo descrito en el numeral 5.9.3
Entradas	▪ **Agile-DISOP** (General)
Salidas	▪ **Agile-DISOP** (Específico)
Roles Responsables:	▪ Ingeniero de Procesos
Nombre de Actividad	Nivel de Proyecto

Descripción	Su especificación corresponde al flujo de trabajo mostrado en la Figura 47: Flujo de Trabajo para el Mejoramiento del Proceso en la sección derecha "Nivel de Proyecto".
Entradas	▪ **Ninguna.**
Salidas	▪ **Ninguna.**
Roles Responsables:	▪ Ingeniero de Procesos y Equipo (i+d) responsable del Proyecto.
Nombre de Actividad	Incorporar estrategias de mejora
Descripción	Consiste en refinar el proceso de desarrollo al reemplazar prácticas actuales por otras más refinadas y maduras que demostraron beneficios durante la aplicación en uno o varios sub-proyectos de la comunidad.
Entradas	▪ **Agile-DISOP** (Versión i)
Salidas	▪ **Agile-DISOP** (Versión i+1)
Roles Responsables:	▪ Ingeniero de Procesos
Nombre de Actividad	Elaborar / Refinar Guía del Proceso
Descripción	Consiste en plasmar las estrategias y prácticas refinadas en la guía global del proceso de desarrollo con el objeto de que los beneficios derivados de las áreas mejoradas sean propagados a todos los equipos (i+d) que conforman la comunidad de desarrollo.
Entradas	▪ **Guía Global del Agile-DISOP** (Versión i)
Salidas	▪ **Guía Global del Agile-DISOP** (Versión i+1)
Roles Responsables:	▪ Ingeniero de Procesos

Tabla 27: Actividades del Nivel de Proceso para el Modelo de Mejora

A continuación, se especifican las actividades relacionadas con el flujo de trabajo del Proyecto:

NIVEL DEL PROYECTO	
Nombre de Actividad	Planear Áreas de Mejora
Descripción	Consiste en determinar sobre que áreas del **Agile-DISOP** se enfatizará para aplicar un enfoque de mejora. Por ejemplo: Para el proyecto actual, se enfocará en estandarizar toda la documentación del proceso o se implementará un modelo para el monitoreo y control de los riesgos. En concreto cada área de mejora se puede enfocar en el refinamiento iterativo e incremental de cada una de las prácticas recomendadas por **Agile-DISOP** numerales 5.4.1 a 5.4.11 y/o las disciplinas especificadas en los numerales 5.9 a 5.14.
Entradas	▪ **Plan de Mejoramiento** (En Blanco)
Salidas	▪ **Plan de Mejoramiento** (Enuncia las áreas de Mejora que son objetivo)
Roles Responsables	▪ Ingeniero de Procesos
Nombre de Actividad	Ejecutar Proyecto
Descripción	▪ La especificación de esta actividad se encuentra en la **Disciplina de Gestión del Proyecto** numeral 5.11.3**,** Figura 42: Flujo de Trabajo para la Gestión del Macro-Proyecto, Figura 43: Flujo de Trabajo para el Modelo de Gestión en un Sub-Proyecto y numeral 5.11.4.
Entradas	▪ **Ninguna**
Salidas	▪ **Sub-Proyecto Finalizado / Sub-Proyecto Cancelado**
Roles Responsables	▪ Director de Sub-Proyecto **i**

	▪ Equipo (i+d) responsable del Sub-Proyecto i
Nombre de Actividad	Ejecutar Estrategias de Mejora
Descripción	Consiste en aplicar las estrategias de mejora sobre las áreas seleccionadas para mejoramiento mientras se está ejecutando el proyecto de desarrollo.
Entradas	▪ **Plan de Mejoramiento** (Enuncia las áreas de Mejora que son objetivo)
Salidas	▪ **Ninguna**
Roles Responsables	▪ Ingeniero de Procesos
Nombre de Actividad	Evaluar Impacto de las Estrategias de Mejora
Descripción	Consiste en la valoración de los impactos (positivos o negativos) derivados de la aplicación de las estrategias sobre las áreas del **Agile-DISOP** seleccionadas para mejoramiento. Esta valoración debe presentar información estadística relacionada con variaciones sobre el esfuerzo, el costo, la calidad, el tiempo entre otros.
Entradas	▪ **Informe de Valoración sobre las Estrategias de Mejora** (En Blanco)
Salidas	▪ **Informe de Valoración sobre las Estrategias de Mejora** (Diligenciado)
Roles Responsables	▪ Ingeniero de Procesos
Nombre de Actividad	Formalización de las Estrategias de Mejora
Descripción	Consiste en la estandarización de las estrategias de mejora con el propósito de permitir su unificación y posterior incorporación al proceso **Agile-DISOP** con el objeto de propagar más adelante los beneficios para toda la comunidad de desarrollo.
Entradas	▪ **Documento Formalización Estrategias de Mejora** (En Blanco)
Salidas	▪ **Documento Formalización Estrategias de Mejora** (Diligenciado)
Roles Responsables	▪ Ingeniero de Procesos

Tabla 28: Actividades del Nivel del Proyecto para el Modelo de Mejora

5.13.5. Artefactos

Los artefactos elaborados, refinados o involucrados durante la ejecución de las actividades de esta disciplina se relacionan en la siguiente tabla:

No.	Nombre	Propósito	Especificado en…
1	Proceso **Agile-DISOP**	Proceso Ágil para Desarrollo en Comunidad.	Anexo 5
2	Guía Global de **Agile-DISOP**	Documentación del Proceso que incluye un esquema para seguirlo.	Anexo 5
3	Plan de Mejoramiento	Determina que áreas se van a mejorar durante un proyecto cualquiera.	No Disponible
4	Informe Valoración sobre las Estrategias de Mejora	Determina que impacto tuvo el hacer realidad una mejora dentro de cualquier proyecto.	No Disponible
5	Documento Formalización de las Estrategias de Mejora	Estandariza las áreas de mejora en nuevas prácticas, nuevos artefactos, nuevos flujos de	No Disponible

No.	Nombre	Propósito	Especificado en…
		trabajo etc.	

Tabla 29: Artefactos de la Disciplina Mejoramiento del Proceso

5.13.6. Arquitectura

Como puede apreciarse en la figura siguiente, la relación entre el proceso Agile-DISOP y los proyectos creados a partir de este, tiene implícita información correspondiente a las áreas de mejora que se proponen de ida para cada proyecto y de vuelta se incorporan al proceso.

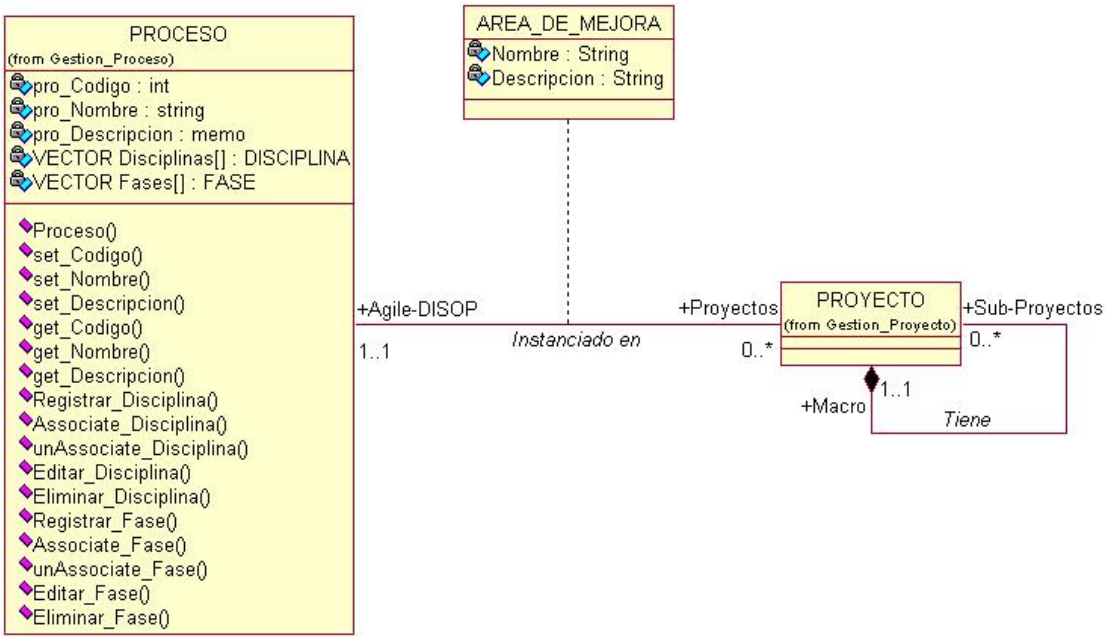

Figura 48: Arquitectura de la Disciplina de Mejoramiento del Proceso

5.13.7. Premisas

- La guía del proceso Agile-DISOP, debe estar publicada en la plataforma de integración u otra plataforma de gestión de contenidos (se recomienda Moodle [8]) con el objeto de facilitar la referencia y consulta por parte de los miembros de la comunidad de desarrollo.

- Es fundamental que exista un modelo de documentación que permita instrumentar esta disciplina. Se recomienda la estructura de directorios del Anexo 5. Allí se muestra la ubicación exacta de los artefactos de esta disciplina en el marco de una iteración para cualquier fase del proceso. La idea es que con cada nueva iteración, el usuario de Agile-DISOP, sólo replicará el sub-directorio Iteración, logrando con esto, alcanzar una estructura de documentación uniforme para todos los proyectos que emprenda la comunidad.

5.14. DISCIPLINA: GESTION DEL CAMBIO DE AGILE-DISOP

5.14.1. Propósito

Esta disciplina es conforme a la práctica sugerida en el numeral 5.4.8 y su objeto fundamental consiste en:

- Garantizar la integridad de todos los artefactos del proyecto a medida que estos evolucionan con el devenir de su ejecución.
- Asegurar la completitud y correctitud de los modelos con el producto en desarrollo.
- Proveer un entorno estable para desarrollar sin conflictos.

La ejecución de esta disciplina para cualquier proyecto de desarrollo en comunidad, involucrará: la identificación, control, gestión y auditoria de los artefactos del proyecto. Controlar el cambio en los artefactos del proyecto, evita los sobre costos debido al caos y la confusión de versiones no controladas y asegura que los artefactos resultantes no generan conflictos entre si, debido a los siguientes inconvenientes:

- **Actualización simultánea sobre los artefactos sin soporte automático de control de versiones.** Cuando dos o más miembros de un equipo, trabajan de forma separada, posiblemente desde ubicaciones geográficamente distantes y hasta de forma asíncrona, sobre el mismo artefacto, los últimos cambios hechos, sobrescribirán a los anteriores, generando conflictos entre los responsables del artefacto y ralentizando el proceso de desarrollo.

- **Notificación de cambios importante no propagada.** Cuando un problema importante es resuelto en artefactos compartidos entre varios desarrolladores sin el soporte de un sistema de gestión del cambio, esta solución no será notificada a todos los interesados.

- **Versiones Múltiples.** La mayoría del software es desarrollado de forma iterativa e incremental, liberando ocasionalmente versiones del mismo. Así, una versión del software puede estar en el entorno de cliente, mientras que otra esta en etapa de pruebas o desarrollo. Si se encuentran problemas en alguna de estas versiones, sus soluciones deben ser propagadas a todas las demás. La confusión puede acarrear costos y trabajo innecesarios si no se cuenta con un sistema que controle los cambios.

5.14.2. Conceptos

Los conceptos involucrados en esta disciplina son: Correctitud, Completitud, Artefacto, Sistema de Gestión del Cambio, Solicitud de Cambio, Orden de Cambio. Para encontrar las definiciones correspondientes a los anteriores conceptos favor referirse al Anexo 3.

5.14.3. Flujo de Trabajo

En el flujo de trabajo de la Figura 49, las solicitudes de cambios sobre las características del producto software en desarrollo van pasando por un proceso

continuo y gestionado de envío refinamiento, revisión, aprobación o rechazo. Si estas solicitudes son aprobadas, entonces el responsable (Gestor del Cambio) deberá planificar el esfuerzo, tiempo y coste que implica integrar el cambio dentro de la iteración actual y finalmente asignar a un responsable que integre el cambio aprobado en la solicitud y dentro de la iteración.

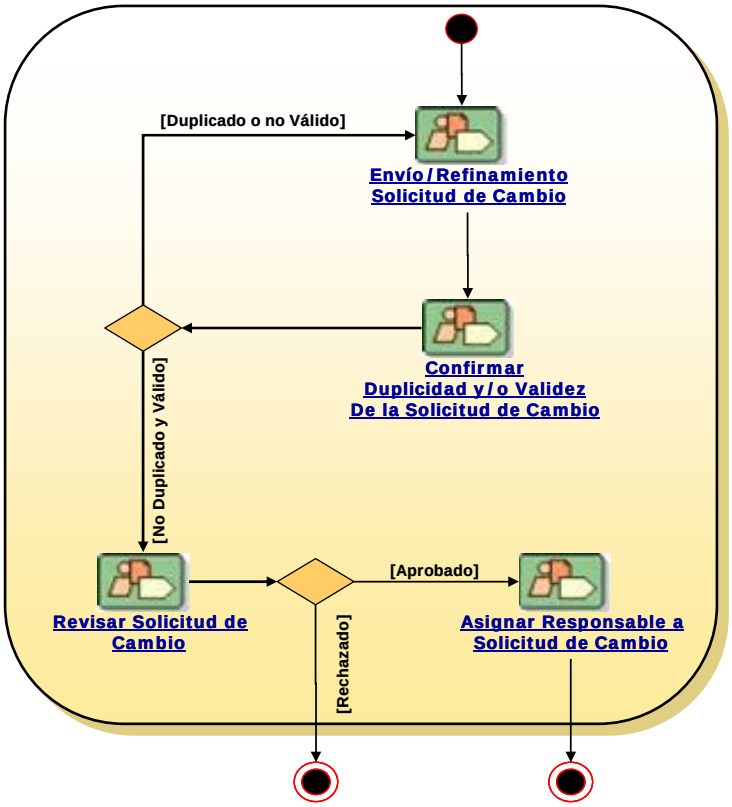

Figura 49: Flujo de Trabajo de la Gestión del Cambio

5.14.4. Actividades

A continuación se especifican las actividades contenidas en el flujo de trabajo de la Figura 49:

Nombre de Actividad	Envío / Refinamiento de una Solicitud de Cambio
Descripción	Esta actividad tiene como propósito el registro de una solicitud de cambio, la cuál puede incluir: ▪ Nuevas características para el sistema o mejora de las existentes. ▪ Corrección o cambio en los requerimientos. ▪ Corrección, refinamiento o cambio en cualquier modelo o documento. Cualquier Rol, puede enviar una solicitud de cambio como actividad cotidiana dentro del ciclo de vida del proyecto.
Entradas	▪ **Solicitud de Cambio** (En Blanco)
Salidas	▪ **Solicitud de Cambio** (Diligenciada)
Roles Responsables	▪ Cualquiera
Nombre de Actividad	Confirmar Duplicidad y/o Validez de la Solicitud de Cambio
Descripción	El propósito de esta actividad consiste en confirmar si una solicitud de cambio está duplicada (ya se ha hecho antes) o si es válida.
Entradas	▪ **Solicitud de Cambio** (Diligenciada)
Salidas	▪ **Solicitud de Cambio** (Diligenciada y Validada)
Roles Responsables	▪ Gestor Control de Cambios
Nombre de Actividad	Revisar Solicitud de Cambio
Descripción	El propósito de esta actividad consiste en determinar si la solicitud de cambio debe ser aceptada o rechazada. En caso de ser aceptada, el Gestor del Control de Cambios valorará la prioridad, esfuerzo, costo y tiempo para determinar si los cambios solicitados caben en el alcance de la actual versión del producto.
Entradas	▪ **Solicitud de Cambio** (Diligenciada y Validada) ▪ **Plan de Iteración**
Salidas	▪ **Solicitud de Cambio** (Diligenciada, Validada y Aceptada ó Rechazada)
Roles Responsables	▪ Gestor Control de Cambios
Nombre de Actividad	Asignar Responsable a Solicitud de Cambio
Descripción	El propósito de esta actividad consiste en acomodar los cambios aprobados tanto al proceso como al producto a medida que se van dando en una iteración mediante la asignación de esta responsabilidad al Director del Proyecto.
Entradas	▪ **Solicitud de Cambio** (Diligenciada, Validada y Aceptada) ▪ **Plan de Iteración**. ▪ **Plan de Proyecto**.
Salidas	▪ **Plan de Iteración** (Ajustado al cambio aprobado) ▪ **Plan de Proyecto** (Ajustado al cambio aprobado) ▪ **Orden de Cambio**
Roles Responsables	▪ Director del Proyecto

Tabla 30: Actividades de la Gestión del Cambio

5.14.5. Artefactos

Los artefactos elaborados, refinados o involucrados durante la ejecución de las actividades de esta disciplina se relacionan en la siguiente tabla:

No.	Nombre	Propósito	Especificado en…
1	Solicitud de Cambio	Registra una solicitud de cambio	Anexo 5
2	Plan de Iteración	Artefacto que estima los recursos, productos, tiempo, esfuerzo y personal para llevar a cabo una iteración.	Anexo 5
3	Plan de Proyecto	Igual que el anterior pero para todo el proyecto.	Anexo 5
4	Orden de Cambio	Emite una orden para realizar un cambio por el que alguien responderá.	Anexo 5

Tabla 31: Artefactos de la Disciplina Gestión del Proyecto

5.14.6. Arquitectura

En la siguiente figura se ilustra cómo cualquier rol puede solicitar un cambio dentro de cualquier iteración. Esta solicitud se estudia y es asignada a cualquier rol para su implementación. Las solicitudes y realizaciones de los cambios, afectan el estado global de la iteración misma, principalmente porque un cambio realizado significa modificación en uno o más artefactos en cualquiera de las disciplinas de Agile-DISOP.

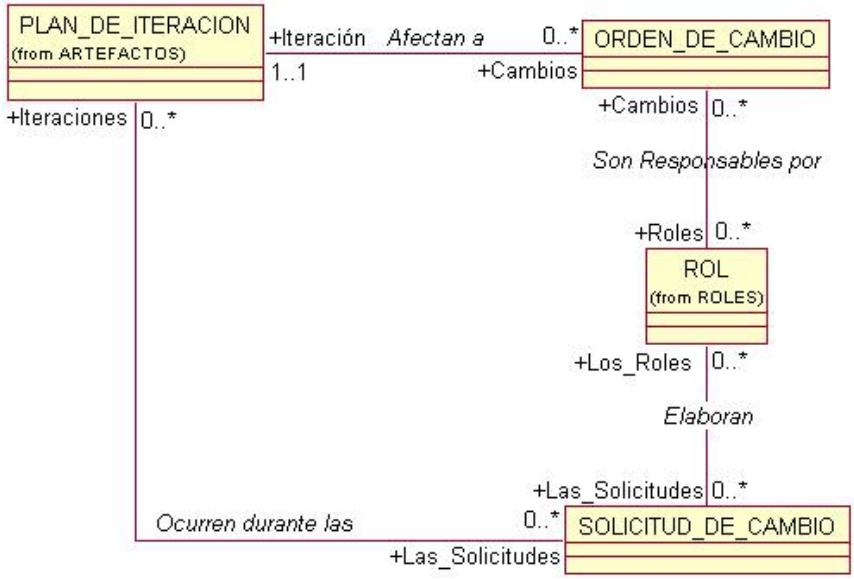

Figura 50: Arquitectura de la Gestión del Cambio

5.14.7. Premisas

- La comunidad de desarrollo debe soportar las actividades derivadas de la gestión del cambio mediante el uso de una herramienta software que automatice el control de las versiones para cada uno de los artefactos generados durante el proyecto por la comunidad de desarrollo. Se recomienda la herramienta de control de versiones CVS (http://www.cvs.org), cuya distribución es libre y funciona sobre web.

5.15. PLATAFORMA DE INTEGRACION PARA LA COMUNIDAD

Según lo recomendado en la práctica 5.4.9 es crucial para soportar la consolidación de la comunidad de desarrollo, la promoción y uso de una herramienta de integración basada en las TIC para mitigar los efectos devastadores que puede ocasionar la dispersión geográfica sobre la gestión, coordinación y control de los equipos de desarrollo y proyectos en comunidad. El proceso de selección de la plataforma orientada hacia la integración para la comunidad cubrió un conjunto de actividades que comprendieron desde el estudio y comparación de algunas herramientas para el soporte a la gerencia de proyectos disponibles en el mercado hasta la selección, configuración y puesta en marcha de la plataforma de integración escogida. A continuación se presenta una exploración de herramientas previa a la selección de la plataforma de integración.

5.15.1. Exploración de Herramientas para la gerencia de proyectos

Esta exploración se concentro en analizar las características más significativas (positivas y negativas) de algunas de las herramientas para la gerencia de proyectos que están disponibles en el mercado. Al final se provee una comparación acerca de las herramientas tratadas con el propósito de determinar un conjunto de criterios que permitan la selección de una herramienta que brinde soporte a la ejecución de proyectos en comunidad.

5.15.1.1. FastTrack

FastTrack es un software para la gerencia de proyectos y su interfaz se muestra en la Figura 51. Este software fue desarrollado por AEC Software, la versión del sistema evaluada fue la FastTrack 6.04 en versión demo cuya única limitación consiste en no poder salvar la información de un proyecto a disco.

Puntos a Favor.

- FastTrack tiene como objetivo, mantener el histórico de proyectos, actividades, tareas y plazos finales. En su nivel más básico, FastTrack automatiza la elaboración de gráficos de Gantt mediante la representación de las actividades como una línea continua cuya longitud esta definida por el inicio y final de la actividad. En caso de que una línea (que representa una actividad) sea alterada en su longitud, los cambios de inicio y final para la actividad correspondiente, serán reflejados automáticamente así como sus datos y costo relativo. Pueden escogerse hasta 17 tipos de barras de actividad para representar información diferente. FastTrack puede mostrar el diagrama de Gantt en una escala de tiempo mayor o menor, por ejemplo: horas, días, semanas, trimestres o años.

- FastTrack provee cuadros de dialogo para especificar la información de las tareas, incluso si el usuario desea, puede crear una jerarquía de tareas con el propósito de representarlas en el grafico de Gantt. De otra parte, la herramienta provee un mecanismo para rastrear el estado de ejecución de cada tarea (no iniciada, en ejecución o finalizada), con el propósito de reflejar la varianza entre la estimación inicial y la ejecución efectiva del proyecto.

- FastTrack ofrece la capacidad de ejecutar “scripts” con el propósito de especificar y automatizar un conjunto de acciones.

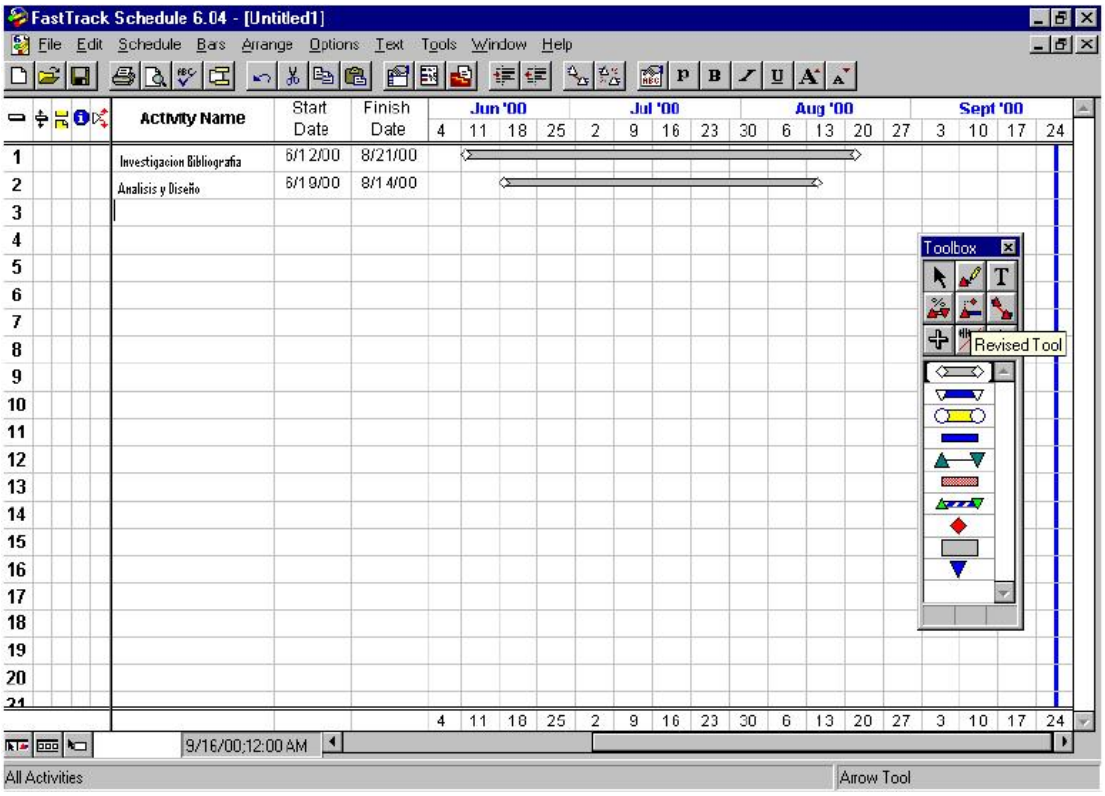

Figura 51: Interfaz de FastTrack

Puntos en Contra.

- El software permite que se creen enlaces incorrectos entre tareas o entre fases del proyecto.

- No brinda soporte a la integración de la comunidad de desarrollo, pues sus servicios solo se limitan al monitoreo y control de un proyecto, no permitiendo la interacción de los miembros de una comunidad de desarrollo a través de una intranet o Internet.

- Al ser software propietario, sus requerimientos de hardware, software y su costo, pueden llegar a ser prohibitivos para algunos usuarios de la comunidad.

- Finalmente debe instalarse en cada computador que haga para de la comunidad, pero sin la facilidad para compartir datos entre ellos, dando lugar a situaciones donde exista información redundante.

5.15.1.2. Task Manager

Task Manager es un software que brinda soporte al gerenciamiento de proyectos y un ejemplo de su interfaz se ilustra en la Figura 52. Este software fue desarrollado por OrbiSoft (http://www.orbisoft.com), la versión estudiada fue la 1.00.729.

Puntos a Favor.

- Task Manager 2000 fue diseñado para ayudar a organizar y administrar todos los trabajos y tareas de forma simple mediante el uso de asistentes.

- De otra parte, Task Manager 2000, permite se utilizado individual o colectivamente, brindando soporte a la capacidad de administrar el personal además de compartir tareas, trabajos y proyectos. Lo anterior es posible, gracias a la disposición de una base de datos localizada en un único computador y disponible a través de la red para todos los equipos, los cuales recuperan información diversa para proyectos en particular.

- Este software permite adquirir una visión rápida de todas las tareas, controlar los plazos finales, controlar de forma automática las cargas de trabajo y tiempo. Además, permite el cálculo automático del costo total del proyecto con base en los costos de cada trabajador asignado al proyecto.

Puntos en Contra.

- Aunque provee una interfaz de fácil manejo, es deficiente en la parte de representación de gráficos. Algunos de ellos no ofrecen la capacidad de

modificarse mediante parámetros, por ejemplo la asignación de recursos, costos, tareas el software solo despliega fechas para los últimos 5 años.

- No ofrece gráficos significativos como PERT o Gantt que son extensamente utilizados en la gerencia de proyectos.
- En ocasiones su desempeño es ralentizado por el cambio entre una ventana y otra.
- Tiene una ayuda en línea para obtener soporte sobre cualquier duda acerca del uso de la aplicación, sin embargo este no estará disponible para usuarios no conectados.

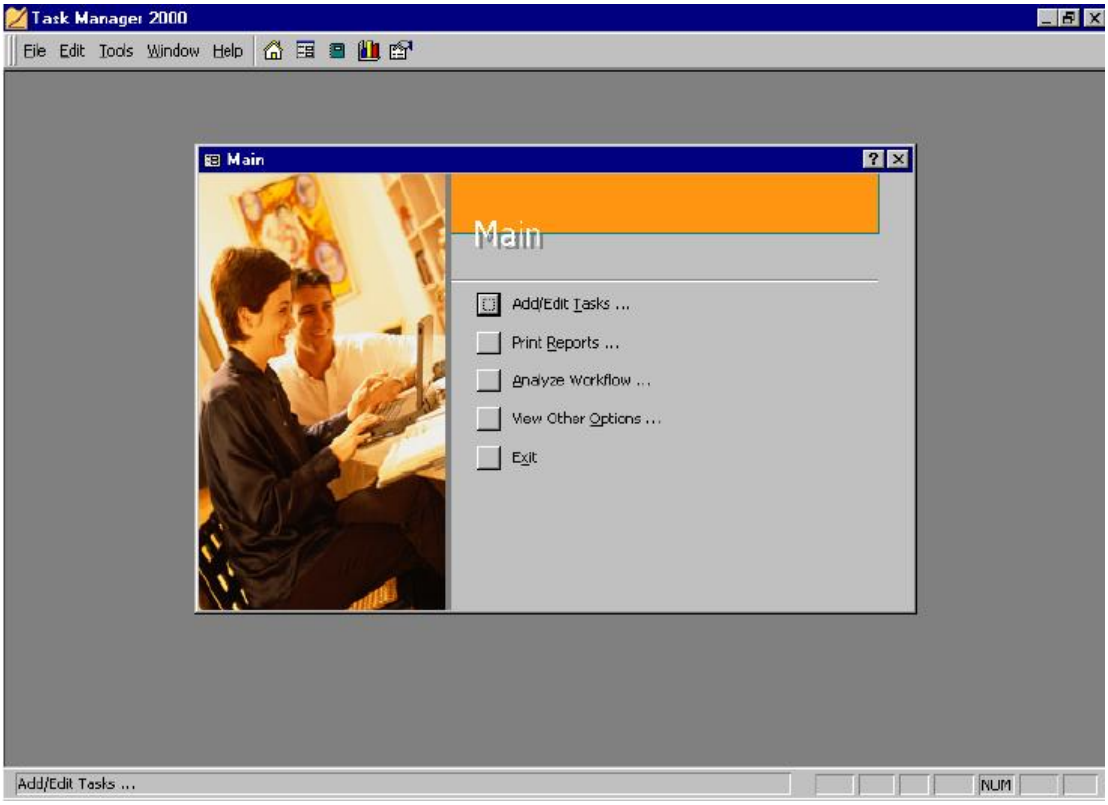

Figura 52: Interfaz de Task Manager

- No brinda soporte a la integración de la comunidad de desarrollo, pues sus servicios solo se limitan al monitoreo y control de un proyecto, no permitiendo la interacción de los miembros de una comunidad de desarrollo a través de una intranet o Internet.

- Al ser software propietario, sus requerimientos de hardware, software y su costo, pueden llegar a ser prohibitivos para algunos usuarios de la comunidad.

5.15.1.3. Delegator

Delegator es mucho más que un software de gerenciamiento de proyectos, cuya interfaz se muestra en la Figura 53. Este software fue desarrollado por Madrigal Soft Tools Inc, (http://www.madrigalsoft.com) con sede en Canadá. La versión demo utilizada fue desarrollada en el 2000 y actualmente se encuentra en la versión 3.0.

Puntos a Favor.

- Delegator fue concebido especialmente para gerentes y demás personas que coordinan y administran las actividades de otras. Delegator ayuda a localizar las diversas tareas asignadas a los miembros del equipo, determinar la carga de trabajo del personal y su desempeño.

- Utiliza gráfico de Gantt con el propósito de visualizar las tareas, y sus responsables con la capacidad de ajustar la escala de tiempo deseada (días, semanas, meses o trimestre).

- Además, se destaca la disponibilidad de una ayuda en línea que soporta el uso correcto de la herramienta.

- Como punto principal se resaltan los tres planos del cronograma que son dispuestos en el grafico de Gantt (ver Figura 53), la barra en amarillo se refiere a la planificación temporal inicial especificada por el gerente para una determinada tarea, la barra roja se refiere a una planificación temporal mas reciente especificada por el gerente, finalmente la barra de color azul, indica la ejecución temporal real realizada por el grupo responsable de la tarea. A través de estos tres planos, es posible realizar un comparativo de la ejecución temporal de las tareas desde su concepción hasta su conclusión.

- Mediante esta herramienta es posible generar informes acerca del desempeño histórico para: personas, equipos y proyectos entre otros. Todos los informes se pueden configurar por parte del usuario, mediante parámetros relativos a las fechas de inicio y finalización.

Puntos en Contra.

- Falta de interactividad con el grafico de Gantt debido a que las tareas dispuestas en el grafico, solo aceptan entrada de datos de forma textual. Además de esto, no permite establecer dependencia entre tareas.

- No brinda soporte a la integración de la comunidad de desarrollo, pues sus servicios solo se limitan al monitoreo y control de un proyecto, no permitiendo la interacción de los miembros de una comunidad de desarrollo a través de una intranet o Internet.

- Al ser software propietario, sus requerimientos de hardware, software y su costo, pueden llegar a ser prohibitivos para algunos usuarios de la comunidad.

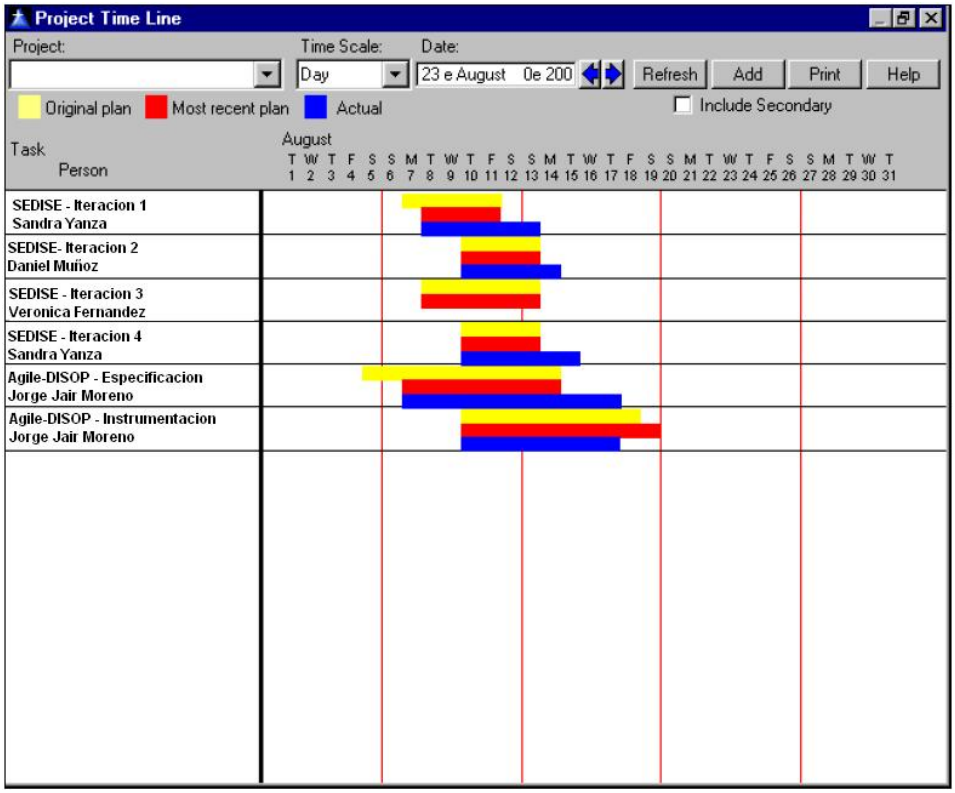

Figura 53: Interfaz Delegator

5.15.1.4. AlexSys Team

Alexsys Team es un sistema de administración de equipos para Windows que organiza el trabajo de un grupo humano dentro de una organización. La interfaz de esta herramienta se muestra en la Figura 54.

Puntos a Favor.

- El equipo puede definir tareas de forma colectiva y auto-organizarse para ejecutarlas. También, el equipo puede detectar cuando una tarea no se esta ejecutando según lo planeado.

- Durante la instalación, se copian dos bases de datos: una en blanco y una de muestra. La base de datos en blanco, permite a los gerentes de la

organización, registrar el trabajo de sus subordinados, mientras que la base de datos de muestra, provee un conjunto de ejemplos que indican las posibilidades de la herramienta.

- La información pertinente a todas las actividades se registra en tablas anexas que posteriormente son referenciadas dentro de la estructura de cada actividad. Además, es posible enviar información de una tarea a otra persona del proyecto mediante el uso del correo electrónico.

- Se destaca la flexibilidad en la configuración del ambiente de trabajo para ajustarse al estilo del usuario.

- Por otra parte, es sencilla la creación y almacenamiento de gráficos (con varios estilos) y reportes con base en cualquier información almacenada en la base de datos del sistema.

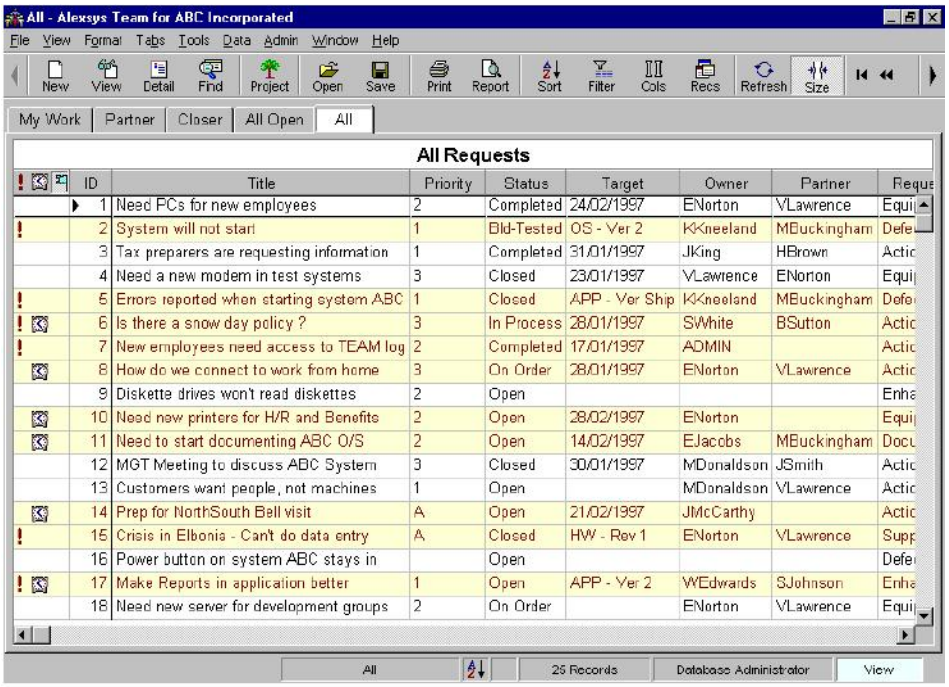

Figura 54: Interfaz de AlexSys Team

Puntos en Contra.

- No existe un grafico (Gantt o PERT) que brinde soporte a la gerencia de tareas y del cronograma, de tal forma que el gerente no puede visualizar la distribución de las tareas con relación al tiempo del proyecto. La herramienta tan solo provee un listado de tareas.

- No brinda soporte a la integración de la comunidad de desarrollo, pues sus servicios solo se limitan al monitoreo y control de un proyecto, no permitiendo la interacción de los miembros de una comunidad de desarrollo a través de una intranet o Internet.

- Al ser software propietario, sus requerimientos de hardware, software y su costo, pueden llegar a ser prohibitivos para algunos usuarios de la comunidad.

5.15.1.5. MS-Project

MS-Project es una herramienta para la gerencia, organización y control de proyectos. Para el análisis se utilizo la versión 2003 sin restricciones. La interfaz de MS-Project se muestra en la Figura 55.

Puntos a Favor.

- El propósito fundamental de MS-Project consiste en apoyar la gerencia de las tareas en función del tiempo. Soporta gráficos de Gantt para la visualización de tareas, las cuales se representan mediante barras (interactivas) cuya longitud esta directamente relacionada con la duración de la tarea. En cada tarea se especifica información relacionada con el inicio y finalización de la misma la cual puede ser accesada directamente sobre las barras graficas de tareas en el diagrama de Gantt.

- También soporta gráficos de PERT con el propósito de visualizar con mayor énfasis en las dependencias entre tareas. Así mismo, el software provee un grafico de recursos el cual permite visualizar la asignación del esfuerzo atribuido a cada recurso del proyecto.

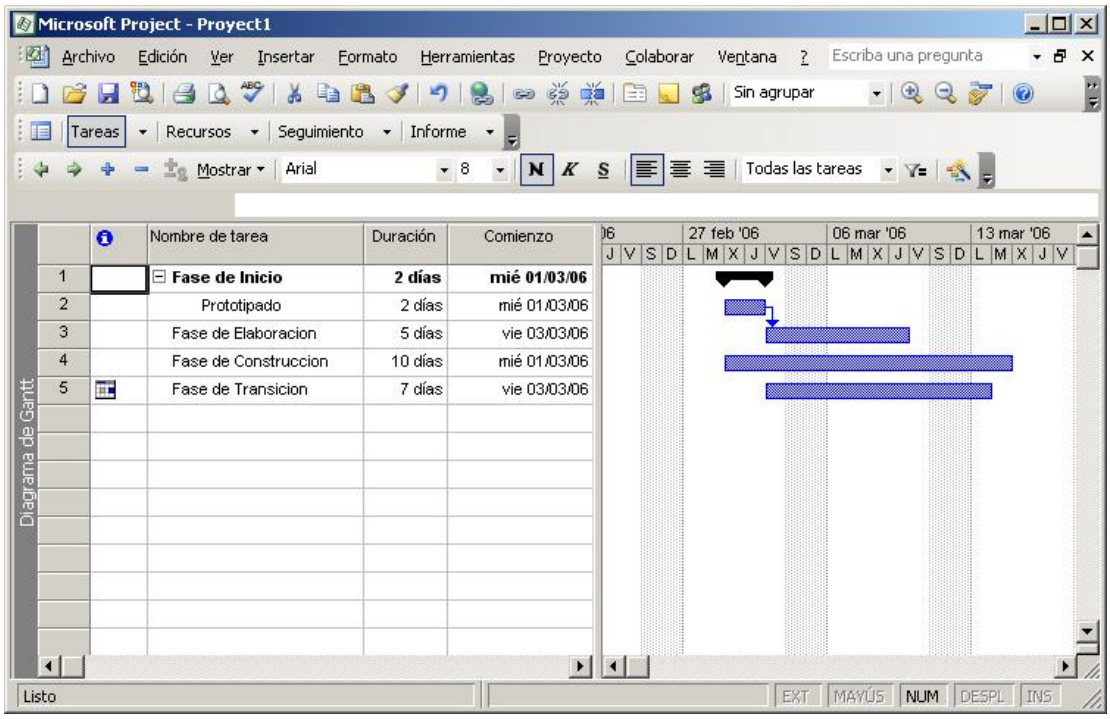

Figura 55: Interfaz de MS-Project

- Antes de que ocurran problemas relativos a conflictos en la planificación temporal, este software implementa una funcionalidad para detectarlos.

- Implementa una línea base que contiene estimaciones originales de costos, recursos y planificación temporal. Al comparar, las estimaciones de la línea base con la ejecución actual, se pueden adaptar la planificación entera del proyecto.

- Se pueden publicar en formato HTML, información de interés para el equipo humano que ejecuta el proyecto. Además se facilita la utilización del e-mail como medio para el envío/recepción de mensajes entre los integrantes del equipo. El asunto de cada mensaje puede variar entre: asignación, aceptación, rechazo, actualización o informe de avance para cualquier tarea.

- Provee flexibilidad en la generación de informes acerca del proyecto. Existen más de veinte (20) tipos de informes.

- Presenta inter-operabilidad con Project Server la cual le permite exportar y compartir información para todo el equipo de desarrollo en el Web.

Puntos en Contra.

- No brinda soporte a la integración de la comunidad de desarrollo, pues carece de servicios de comunicación y coordinación tales como: calendarios personales y de grupo, foros de debate, help-desk entre otros.

- Al ser software propietario, sus requerimientos de hardware, software y su costo, pueden llegar a ser prohibitivos para algunos usuarios de la comunidad. Finalmente para acceder a los servicios que le permiten compartir información a través de la Web, es necesario instalar otras herramientas propietarias que aumentan todavía mas el costo y complejidad del software de la plataforma de integración para cada equipo (i+d) de la comunidad de desarrollo.

5.15.1.6. SuperProject

SuperProject es una herramienta software concebida para la gerencia de proyectos, producida por Computer Associates, cuya interfaz puede observarse en la Figura 56. El software utilizado fue la versión 4.0 sin ninguna restricción. Esta herramienta se compone de dos módulos. El primero, denominado Application Program Interface

se concentra en la gestión de proyectos. El segundo, denominado CARealizer ofrece un lenguaje para la creación de macros.

Puntos a Favor.

- Este software permite la gestión de recursos, costo y tiempo de los proyectos, mediante gráficos de Gantt y/o PERT. Ofrece la posibilidad de la generación de proyectos a partir de plantillas. SuperProject es bastante versátil para la elaboración de informes.

- Además, permite ajustar la unidad de tiempo a las necesidades del gerente de proyecto. Además, provee diversas perspectivas por ejemplo: la utilización de recursos, costos involucrados, especificación de tareas entre otras. Permite incluir sub-proyectos o establecer enlaces entre proyectos asociados externos.

Puntos en Contra.

- No brinda soporte a la integración de la comunidad de desarrollo, pues sus servicios solo se limitan al monitoreo y control de un proyecto, no permitiendo la interacción de los miembros de una comunidad de desarrollo a través de una intranet o Internet.

- Al ser software propietario, sus requerimientos de hardware, software y su costo, pueden llegar a ser prohibitivos para algunos usuarios de la comunidad.

- La capacidad de definir proyectos y sub-proyectos solo esta disponible para archivos sobre la misma maquina.

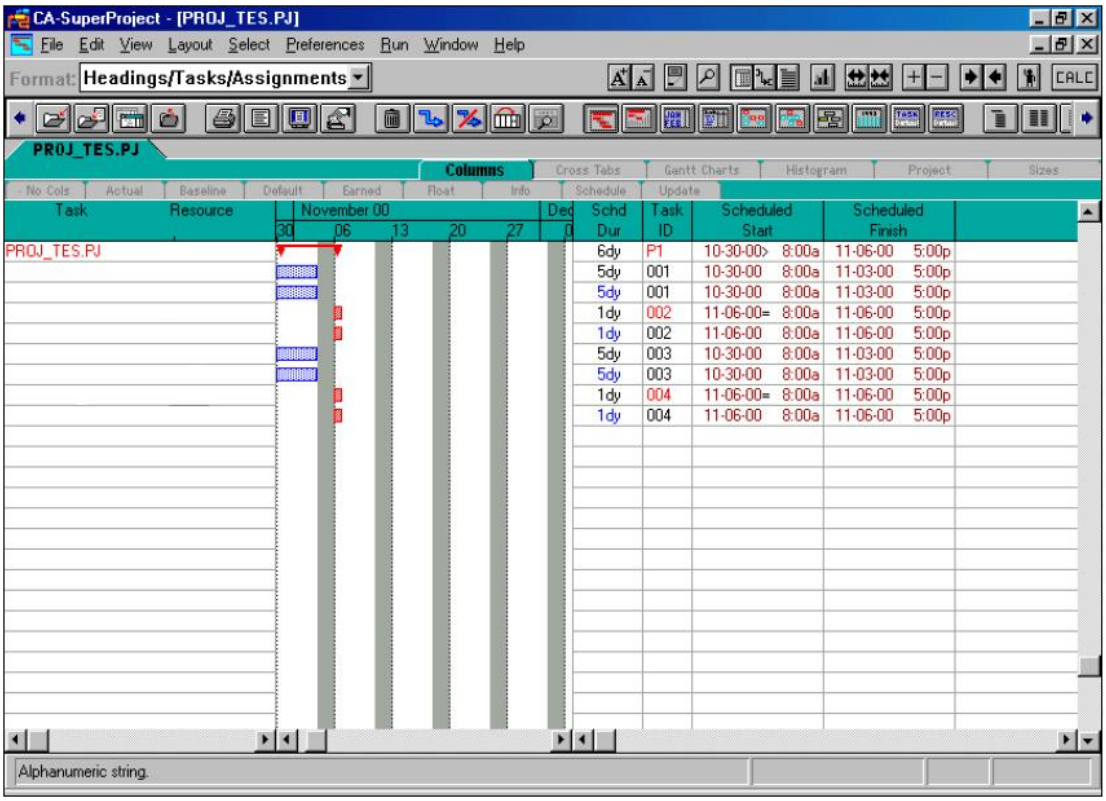

Figura 56: Interfaz de SuperProject

5.15.1.7. SourceForge. NET

SourceForge.net es la plataforma web cuya interfaz puede observarse en la Figura 57. Esta plataforma reúne a la comunidad más grande del mundo para desarrollo de software libre.

Puntos a Favor.

- Actualmente soporta más de 100.000 proyectos y sobrepasa el millón de usuarios, proporcionando una plataforma de servicios para la gestión de proyectos, de equipos y código.

- Esta plataforma provee hospedaje gratuito para proyectos de desarrollo de código abierto, el cuál tiene como esencia, la rápida creación de soluciones bajo un ambiente colaborativo que comparten los desarrolladores y usuarios finales, los cuáles buscan la calidad y viabilidad del software a largo plazo.

Puntos en Contra.

- Sin embargo, el proceso de registro y adquisición de cuentas de usuario resulto complicado y poco confiable como para ser extendido a la comunidad con buena aceptación.

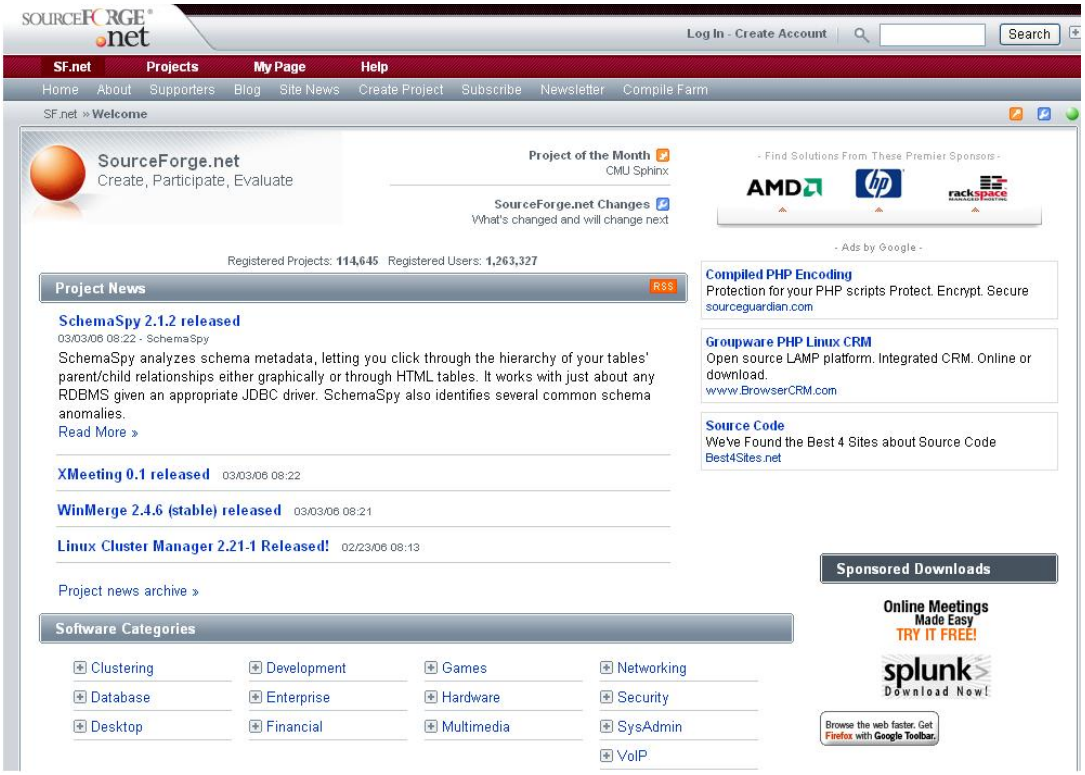

Figura 57: Interfaz de Source Forge

5.15.1.8. PHProjekt

La interfaz de esta plataforma de código abierto puede observarse en la Figura 58, la cuál permite la gestión de proyectos a través de la Web, orientada hacia equipos que pueden estar geográficamente distribuidos conformando una comunidad.

Puntos a Favor.

- Ofrece los servicios de: Calendario, Base de Datos de Contactos, Foro, Archivos (descarga y publicación), proyectos (diagrama de gantt y filtros), timecard (para administrar presupuestos), notas, RTS (Sistema de rastreo de solicitudes), Correo y Administración. La versión utilizada fue la 4.2, la cuál posee la capacidad de integrar la mayoría de componentes.

Puntos en Contra.

- Su uso no resulto sencillo y la estabilidad de la herramienta no resulto satisfactoria debido a las tremendas diferencias de compatibilidad de la versión 4.2 y sus sucesores.

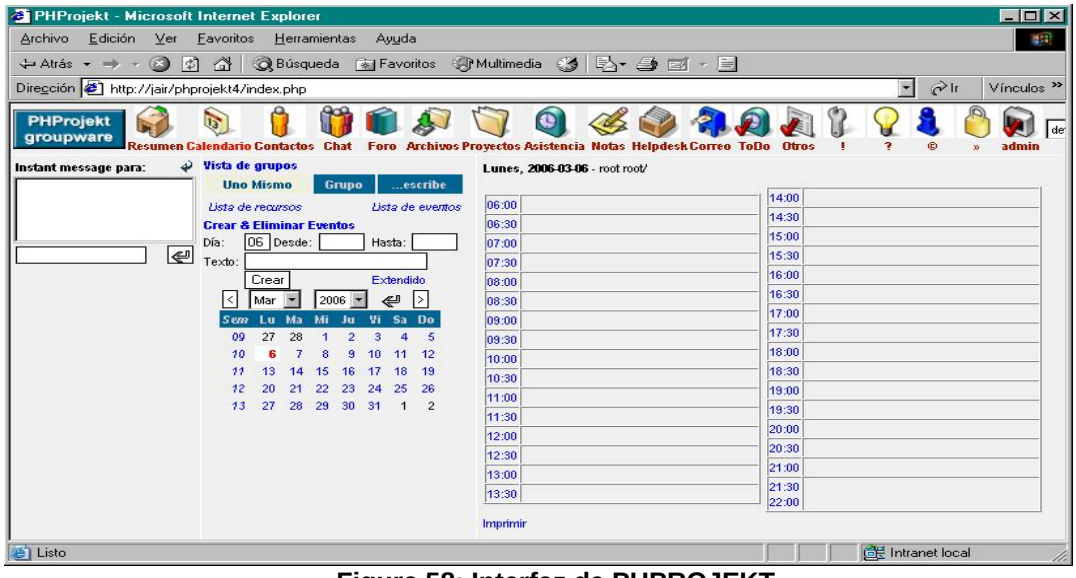

Figura 58: Interfaz de PHPROJEKT

5.15.1.9. DOTProject

Esta plataforma cuya interfaz puede observarse en la Figura 59, permite la gestión de proyectos que nace como alternativa a los costosos productos de Microsoft y otras herramientas disponibles en el mercado.

Puntos a Favor.

- Interfaz de usuario simple y consistente, funcionalidad para la gestión de proyectos, libre uso-acceso y de código abierto. Algunos servicios de la plataforma incluyen: Gestión de Usuarios, E-mail, Gestión de información de clientes y compañías, listado de proyectos, listado jerárquico de tareas, Repositorio de Archivos, Lista de Contactos, Calendario, Foro, Acceso restringido a los recursos.

Puntos en Contra.

- No implementa gráfico de **PERT.**

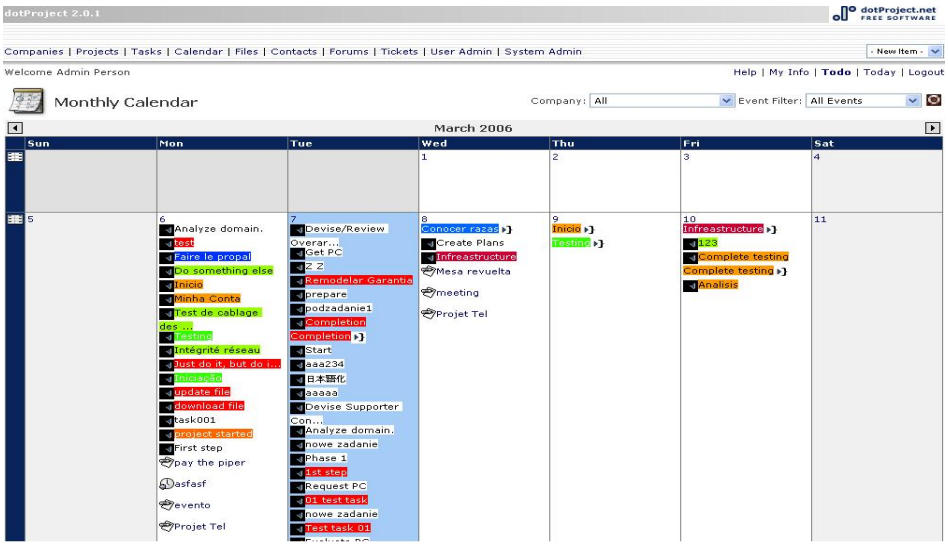

Figura 59: Interfaz de DOTProject

5.15.2. Conclusiones del análisis de las herramientas disponibles

- Se constato que la mayoría de estas herramientas trabaja con los gráficos de Gantt, el grafico de PERT se implementa en menor escala.

- La capacidad de especificar una línea base, es ofrecida por apenas tres herramientas: **MS-Project, DotProject y Delegator**.

- Respecto a los informes, se destacan **MS-Project y Alexys Team** que ofrecen a sus usuarios una gran variedad y flexibilidad.

- Si bien cada herramienta ofrece ayuda en línea para asistir al usuario en el uso de la misma, se destaca **MS-Project** la cual provee algunos tópicos especiales respecto de la gestión de proyectos.

- Con relación al soporte del trabajo en grupo bajo un ambiente de dispersión geográfica, apenas cuatro (4) herramientas ofrecen algún soporte. **MS-Project** permite la exportación y publicación de información relacionada con uno o varios proyectos; además la comunicación entre los miembros del equipo mediante la automatización del correo electrónico. Sin embargo requiere la interoperabilidad de herramientas como **Project Server e Internet Information Server**, instaladas sobre una plataforma de servidor Windows. No ofrece servicios de comunicación como los foros, calendarios personales o grupales, help-desk, mensajes instantáneos, Chat, ni repositorio de documentos controlado por versiones.

- Desde el punto de vista de costo, interoperabilidad y disponibilidad en Internet solo tres (3) plataformas **SOURCEFORGE.Net, PHPROJEKT y DOTProject** ofrecen una integración entre la gestión de proyectos y la coordinación y comunicación de equipos geográficamente dispersos.

- El uso de **PHProjekt** resulto un poco complejo y su desempeño un tanto inestable, de otra parte, la plataforma **SOURCEFORGE**.**Net**, resulto ser la más completa de todas y especialmente enfocada hacia el desarrollo de proyectos de software en comunidad, sin embargo el proceso de registro no resultó confiable debido a que las contraseñas asignadas no funcionaban a la hora de realizar el inicio de sesión en la plataforma, esto preocupo al autor del presente trabajo, fundamentalmente porque el acceso a la comunidad de cualquiera de sus miembros no importando su ubicación geográfica o nivel de habilidad, no debe presentar contratiempos ni complicaciones. De otra parte la herramienta **DOTProject** resulto bastante adecuada para soportar la gestión de proyectos, coordinación y comunicación de los equipos (i+d), no presento complicaciones especiales y en su lugar resulto bastante amigable.

En conclusión, se utilizará la herramienta **DOTProject** para la integración de la comunidad. Esta herramienta estará publicada en el dominio http://pis.unicauca.edu.co/sedise/. La cuenta de acceso del administrador es:

- **Nombre de Usuario:** admin
- **Contraseña:** syncmaster793s

6. PROPUESTA: DESARROLLO EN COMUNIDAD DE UN ESMS-MI

La redacción de este capitulo esta orientada a la satisfacción del tercer objetivo general de este trabajo el cual propone:

- *"Elaborar una propuesta ante COLCIENCIAS, que corresponda al diseño de un plan de proyecto (I+D) (de mediano plazo) para el ESMS de modelos integrados apoyado en el proceso distribuido y soportado en la arquitectura base diseñada, como alternativa de continuidad."*

Al respecto, el grupo GTI (Grupo de Investigación en Tecnologías de la Información) de la universidad del Cauca, al cual pertenece el autor de esta monografía, se involucró en una convocatoria relacionada con los centros de excelencia promovida por COLCIENCIAS, en ella, se propone la armonización de un proyecto macro, planteado por al menos tres (3) grupos de investigación que deseen conformar un Centro de Investigación de Excelencia para trabajo conjunto durante los próximos cinco (5) años. COLCIENCIAS ha planteado la convocatoria en los siguientes términos:

- *"Una red nacional de grupos de investigación del más alto nivel, articulada alrededor de un programa común de trabajo en un área científica y tecnológica considerada como estratégica para el país. Cada uno de los grupos que hagan parte de un Centro de Excelencia deben, además de estar reconocidos o en proceso de reconocimiento al 2005, desarrollar investigación de frontera en permanente contacto con entidades pares internacionales, apoyar la formación de recursos humanos en los niveles de maestría y doctorado, transferir el conocimiento generado al sector productivo, presentar los resultados de su trabajo en publicaciones internacionales indexadas y estar comprometidos en los procesos de protección de la propiedad intelectual y el patentamiento. En el marco de esta convocatoria, podrán participar tres (3) o más Grupos de Investigación*

Científica o Tecnológica del país reconocidos por COLCIENCIAS, en forma conjunta a través de la conformación, por parte de las entidades o instituciones que los avalan (Universidades, Institutos, Centros de Investigación y Centros de desarrollo Tecnológico), de consorcio o unión temporal para la adjudicación, celebración y ejecución del contrato objeto de la presente convocatoria. Las entidades o instituciones que avalen a los Grupos de Investigación Científica o Tecnológica participantes, deberán estar habilitadas jurídicamente para participar en convocatorias, invitaciones o concursos y para celebrar contratos con COLCIENCIAS. Para los efectos de la presente convocatoria se entenderá como Grupo de Investigación Científica o Tecnológica, el conjunto de personas que se reúnen para realizar investigación en una temática dada, formulan uno o varios problemas de su interés, trazan un plan estratégico de largo o mediano plazo para trabajar en él y producen unos resultados de conocimiento sobre el tema en cuestión. Un grupo existe siempre y cuando demuestre producción de resultados tangibles y verificables fruto de proyectos y de otras actividades de investigación convenientemente expresadas en un plan de acción (proyectos) debidamente formalizado."

Al respecto, el autor del presente trabajo, ha realizado una propuesta que encaja en el macro proyecto planteado por GTI[4], denominado CEDUSOC[73], elaborado con motivo de esta convocatoria. Con el propósito de facilitar la lectura de este documento, se ha dispuesto el Anexo 7 cuyo contenido consiste en un resumen de la visión del Centro de Excelencia propuesto por el GTI [4], sus proyectos, investigadores y grupos, al mismo tiempo que exhibe la propuesta para el desarrollo de un Entorno de Soporte a la Ingeniería del Software en Comunidad, denominado SEDISE (en inglés Support Environment for Distributed Software

[73] MODELO DE INTEGRACIÓN DE LAS TIC AL DESARROLLO EDUCATIVO COLOMBIANO EN EL MARCO DE LA SOCIEDAD DEL CONOCIMIENTO

Engineering), cuyo propósito fundamental consistirá en apoyar el trabajo de la comunidad nacional de dinámica de sistemas y pensamiento sistémico, durante el desarrollo del ESMS-MI, entre otros resultados. Si se desea hallar más información detallada acerca de la convocatoria de “Centros de Excelencia” de COLCIENCIAS en la cuál participó GTI, por favor abra el archivo Convocatoria_Centros_de_Excelencia.pdf.

7. CONCLUSIONES Y RECOMENDACIONES.

La culminación del presente trabajo exigió de parte del autor un total de **1151** (Horas/Hombre), de las cuáles **200** (Horas/Hombre) se invirtieron en la redacción de la monografía y la preparación de los anexos. Por otra parte, satisfacer el primer objetivo general de esta tesis, requirió un esfuerzo **511** (Horas/Hombre) ó 44,40% del total, mientras que alcanzar la satisfacción del segundo objetivo general, demandó **420** (Horas/Hombre), lo cuál representa un 36,49% del esfuerzo total del proyecto, por último con un esfuerzo de **20** (Horas/Hombre) representando el 1,74% del total se logró dar cumplimiento al tercer objetivo general. Esta reflexión está motivada por el principio de la ingeniería del software que promueve el mantenimiento de la memoria del proyecto[74], en especial del esfuerzo requerido para alcanzar determinados productos en el marco de un proyecto cualquiera. Por esta razón, la especificación de las conclusiones y recomendaciones que se hacen a cada objetivo general del proyecto, estarán acompañadas del esfuerzo requerido para satisfacerlos. A continuación se enuncian las conclusiones y recomendaciones por cada objetivo general:

En primer lugar, para alcanzar los resultados que corresponden al objetivo general que plantea:

- "*Elaborar los requerimientos funcionales y diseñar la arquitectura base, para un ESMS de modelos integrados, cuyo punto de partida consiste en las arquitecturas software de las herramientas desarrolladas por el grupo SIMON[14] de la escuela de ingeniería de sistemas de la UIS[19].*"

Se hizo necesario realizar una investigación exhaustiva sobre *"Análisis y Evaluación Arquitectónica"*, la cuál, arrojó una metodología de diez (10) etapas y la

[74] Que posteriormente será utilizada en otros proyectos para mejorar la calidad de la estimación del esfuerzo requerido.

armonización de un modelo de métricas para la arquitectura, que exigió alrededor de **115** (Horas/Hombre). Es importante destacar que mediante la metodología concebida, se valoró la calidad arquitectónica de una herramienta real[75], desarrollada desde hace más de doce (12) años al interior del grupo (i+d) SIMON de la UIS. Dicha herramienta con un tamaño de 59.277 líneas de código[76] fue valorada mediante un modelo de hasta 38 métricas propuestas por diversos autores y agrupadas por atributos de *"tamaño", "herencia", "encapsulamiento" y "complejidad"*, también, se aplicaron 4 heurísticas de diseño arquitectónico para determinar problemas relacionados con la *"Completitud", "Correctitud", "Nombrado y Estilo"*. Esta valoración implicó un esfuerzo de **200** (horas/hombre) arrojando un total de 31.503 datos dispuestos en 2.184 registros que se organizaron en 20 aspectos de la herramienta. Por otro lado, con un esfuerzo de **37** (horas/hombre) se encontraron hasta 4.407 problemas arquitectónicos de los cuáles 4.179 problemas se resuelven en la nueva arquitectura del ESMS-MI, quedando 228 problemas que requieren de la atención de los desarrolladores aún sin resolver. Más adelante y luego de cuatro (4) iteraciones y **159** (horas/hombre), la arquitectura resultante con un total de: 6 Diagramas de Casos de Uso, 33 Diagramas de Paquetes, 239 Diagramas de Clases, 1 Diagrama de Componentes, 77 Casos de Uso, 448 clases y 245 paquetes, refleja el espíritu inicial de los desarrolladores de EVOLUCION 3.5 organizando todos los aspectos de la herramienta de forma que el código fuente y el modelo se correspondan mutuamente sin distorsiones favoreciendo la completitud y correctitud de la arquitectura del ESMS-MI. Además el nuevo modelo arquitectónico propone en primer lugar, un conjunto de requisitos funcionales y no funcionales para extender la funcionalidad de EVOLUCION 3.5 hacia el nuevo ESMS-MI, además de algunas mejoras para organizar aún más los elementos de software en el marco de

---

[75] "EVOLUCIÓN 3.5. HERRAMIENTA SOFTWARE PARA EL MODELAMIENTO Y SIMULACIÓN CON DINÁMICA DE SISTEMAS".

[76] En lenguaje **DELPHI** de Borland.

una arquitectura por capas[77] la cuál permitirá involucrar algunos patrones de diseño que se estarían violando en la antigua arquitectura.

Finalmente, se hizo evidente la escasez de herramientas que permitieran realizar de forma automática la totalidad del análisis y evaluación arquitectónicos. Solamente se encontraron tres herramientas[78] semi-adecuadas para tal propósito. Sin embargo se logró aprovechar la combinación estas herramientas mediante técnicas automáticas, semi-automáticas y manuales para cubrir todos los aspectos deseados del análisis y evaluación arquitectónicos. En consecuencia, se encuentra aquí un interesante espacio para la investigación, innovación y desarrollo de modelos de evaluación arquitectónica que en el marco de una tesis de maestría posterior, facilitarían las condiciones idóneas para culminar plenamente el análisis cuantitativo y cualitativo de los datos de la evaluación, garantizando la continuidad y crecimiento de los resultados de este trabajo de tesis.

En segundo lugar, para alcanzar los resultados que corresponden al objetivo general que plantea:

- "*Elaborar una instanciación de un proceso adecuado para el desarrollo distribuido de software que favorezca las condiciones mínimas para unificar e integrar la comunicación y gestión durante la ejecución de un proyecto software orientado a entregar un "Entorno de Modelamiento-simulación de modelos integrados", liderado por el grupo SIMON."*

Durante **10** (Horas/Hombre) se realizó una investigación exhaustiva acerca de los retos relacionados con el desarrollo en comunidad de tal forma que se pudiesen armonizar algunos requisitos fundamentales que el proceso propuesto alcanzaría a

---

[77] Capa de Presentación (Implementa la lógica liviana de la Interfaz Gráfica de Usuario), Capa de Dominio (Implementa la Lógica Gruesa del Negocio) y Capa de Servicios (Implementa la lógica que puede re-utilizarse entre aplicaciones de la misma familia).

[78] ESS-Model, SDMetrics y Ms-Excel.

través de sucesivos niveles de madurez, al final se determinaron para **Agile-DISOP**, un total de 24 requisitos dispuestos en 6 categorías. Así mismo, durante **6** (Horas/Hombre) se rescataron once (11) valiosas prácticas de la ingeniería del software que conformaron los postulados de **Agile-DISOP** en materia de desarrollo comunitario y de otra parte en el Anexo 6, se advierte sobre treinta (30) malas prácticas que entorpecen el desarrollo de cualquier proyecto, lo cuál exigió **4** (Horas/Hombre), por último con un esfuerzo de **19** (Horas/Hombre), se recopilaron 110 riesgos distribuidos en doce (12) categorías relacionadas con la gestión de proyectos informáticos.

Por otra parte, se determino la arquitectura misma del proceso en términos de seis (6) disciplinas de la ingeniería del software y la gestión de proyectos en el marco de una comunidad de desarrollo y se estableció la metodología (Ver numeral 5.6) para especificar todos los componentes del proceso **Agile-DISOP** abarcando un total de **152** (Horas/Hombre). Las **229** (Horas/Hombre) restantes se relacionan con: resultados de investigación sobre las prácticas empresariales de PYMES en el sur-occidente colombiano, la divulgación del proyecto en una revista y dos eventos (uno nacional y uno internacional) y finalmente el emprendimiento y culminación de un trabajo de pregrado[79] que beneficiaría a dos (2) estudiantes del programa de ingeniería de sistemas de la UNICAUCA. Los resultados de este último implicaron para los estudiantes un esfuerzo de 920 (Horas/Hombre), las cuáles sólo 120 (Horas/Hombre) se cuantifican por concepto de la gestión del director. Todos los productos relacionados a este trabajo de pregrado, se incluyen en formato digital en el disco compacto que acompaña a este documento.

Finalmente, aunque los resultados entregados son lo suficientemente completos para ser aprovechados por una comunidad de desarrollo, se torna interesante y

---

[79] Denominado “CONSTRUCCIÓN DE UNA HERRAMIENTA SOFTWARE PARA SOPORTAR UN PROCESO DISTRIBUIDO DE DESARROLLO UTILIZADO POR UNA COMUNIDAD (I+D)”. Universidad del Cauca. Septiembre de 2005.

además recomendable que mediante el mecanismo dispuesto por **Agile-DISOP** explicado en el numeral 5.13. DISCIPLINA: MEJORAMIENTO de Agile-DISOP, se incremente de forma continua y sostenida la madurez de: practicas, disciplinas, actividades, roles y artefactos del proceso. Por otra parte y aprovechando el prototipo entregado por el trabajo de pregrado mencionado anteriormente, puede encontrarse muy interesante una propuesta de tesis de maestría que contemple el desarrollo de una plataforma específica para una comunidad de que utilice no sólo Agile-DISOP, sino cualquier otro proceso de desarrollo.

La estrategia de continuidad para los resultados de esta tesis de maestría consiste en lograr la atención y concentración de una masa crítica de grupos (i+d) interesados en desarrollar sobre la recién abierta arquitectura de EVOLUCION en el ambiente de la plataforma de integración propuesta en el numeral 5.15.2. Al respecto, el autor del presente trabajo ya realizó un ejercicio co-lateral al proyecto que consistió en la instalación, promoción, puesta en marcha y soporte de una comunidad académica sobre la plataforma MOODLE[80], la cuál alcanzo un crecimiento en seis (6) meses de cero (0) a 1640 miembros. Cinco (5) lecciones importantes se destacan para tener en cuenta al momento de emprender la concentración de una comunidad de desarrollo:

1. Los aspectos técnicos de la plataforma software de la comunidad no deben ser evidentes ni mucho menos entorpecer la interacción del usuario al usar los servicios que provee la plataforma. Debe existir un equipo humano trabajando en la sombra para asumir el desgaste técnico relacionado con el soporte de la comunidad. Este aspecto disminuye la posibilidad de frustración y deserción masiva de los usuarios de la comunidad.

[80] Publicada en http:://www.pis.unicauca.edu.co/moodle/

2. El usuario de la comunidad debe percibir conscientemente los beneficios personales que la comunidad le ofrece. Este aspecto ayuda a construir la fidelidad del usuario.

3. La comunidad debe ofrecer permanentemente novedades, recursos e información que sean útiles y efectivas para sus miembros. Se recomienda ir siempre un paso adelante, no solamente dándole al usuario lo que quiere, sino también sorprendiéndolo con lo que no espera.

4. Es clave aliarse con usuarios emprendedores y entusiastas de la comunidad, especialmente si estos se hallan en ubicaciones geográficas lejanas. Esto se debe a que un usuario entusiasta, como ejemplo viviente, promoverá en sus más cercanos los beneficios de agregarse y participar en la comunidad, derrumbando en parte la resistencia de aceptar un paradigma de cooperación diferente.

5. Por último, es vital que al principio de la puesta en marcha de una comunidad, no se sature al usuario con una funcionalidad sobre-sofisticada que pueda aturdirlo o confundirlo causándole frustración y deseo de retirarse. Es fundamental que de forma periódica se capacite a los miembros de la comunidad en el uso correcto de los servicios que esta ofrece, siempre con ejemplos sencillos y en un ambiente informal de mucha cordialidad y respeto hacia el usuario.

En tercer lugar, para alcanzar los resultados que corresponden al objetivo general que plantea:

- "*Elaborar una propuesta ante COLCIENCIAS, que corresponda al diseño de un plan de proyecto (I+D) (de mediano plazo) para el ESMS de modelos integrados apoyado en el proceso distribuido y soportado en la arquitectura base diseñada, como alternativa de continuidad.*"

Se elaboró una propuesta de diez (10) páginas para el desarrollo de una plataforma denominada SEDISE[81] que publicaría la arquitectura del ESMS-MI y apoyaría a SIMON además de otros grupos (i+d) en el desarrollo comunitario de nuevos componentes para la representación de conocimiento en otros enfoques además de la dinámica de sistemas. Esta propuesta fue incluida en el documento "MODELO DE INTEGRACIÓN DE LAS TIC AL DESARROLLO EDUCATIVO COLOMBIANO EN EL MARCO DE LA SOCIEDAD DEL CONOCIMIENTO" (Anexo 7), presentado por el grupo GTI[4]de la UNICAUCA [18]. La propuesta alcanzó a ser evaluada por los pares internacionales que visitaron al GTI en las instalaciones de la UNICAUCA durante 2006, sin embargo recientemente se conoció que no fue aprobada. Se recomienda continuar con la participación en convocatorias a proyectos con el objeto de financiar la sostenibilidad de los resultados del presente trabajo de investigación.

Finalmente,

- **H1:** ¿Es posible diseñar proyectos de desarrollo para comunidades geográficamente dispersas de (I+D) cuyo propósito sea entregar soluciones software a la comunidad académico-investigativa e industrial que compitan en funcionalidad-costo-beneficio con software propietario disponible en el mercado?

- **H2**: ¿Crear las condiciones mínimas para favorecer la consolidación de una comunidad de desarrollo distribuido como estrategia para enfrentar problemas particulares de la academia, la investigación y/o la industria en el marco nacional, representa una alternativa seria al desarrollo tradicional de software propietario?

---

[81] **S**upport **E**nvironment for **DI**stributed **S**oftware **E**ngineering

- **Conclusión.** A través del desarrollo del presente trabajo de tesis el autor ha vivenciado la notoria tendencia que existe actualmente hacia el desarrollo en comunidad. De hecho, a un clic de distancia pueden encontrarse modelos de desarrollo en comunidad muy exitosos, los cuáles validan la viabilidad y competencia de los productos software desarrollados. Al respecto y tomando lectura de las convocatorias de COLCIENCIAS y otros organismos a Centros de Excelencia, es evidente que cada vez se toma más percepción y conciencia acerca de la comunidad (i+d) como un macro-organismo viable e incluso necesario que aborda realidades cada vez más complejas. El desarrollo de software en comunidad soportado por las TIC´s es una oportunidad que justifica y posibilita el diseño de proyectos de contexto e impacto nacionales, competitivos internacionalmente. Aquí lo fundamental consiste en contar por una parte en el liderazgo de un grupo (i+d), una arquitectura de software abierta con potencial y en una comunidad interesada en desarrollar sobre esta arquitectura. Estas condiciones están dadas en el preciso instante de culminación del presente trabajo de investigación.

BIBLIOGRAFÍA

Artículos y Libros

ABRAHAMSSON, P. Salo, RONKAINEN, J. WARSTA, J. “Agile software development methods Review and analysis". http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf. VTT Publications. Año 2002.

ABREU, F y MELO, W. “Evaluating the impact of object oriented design on software quality”. Proceedings of 3rd international software metrics symp. Año 1996.

AMBLER Scott. ”Bridging the Distance”. Software Development. http://www.sdmagazine.com/documents/s=826/sdm0209i/ . Año 2002.

ANDRADE, Hugo; DYNER, Isaac; ESPINOSA, Angela; LÓPEZ, Hernán; SOTAQUIRA, Ricardo. Pensamiento Sistémico: Diversidad en búsqueda de Unidad. Ediciones Universidad Industrial de Santander, Bucaramanga, Colombia. Año 2001.

ARTHUR James D. An Examination of Environment Design and the Applicability of Software Engineering Requirements to the standard simulation Environment. Proceedings of the Winter Simulation Conference. Año 1994.

BARBACCI, et al., Quality Attributes, (CMU/SEI-95-TR-021). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University. http://www.sei.cmu.edu/publications/documents/95.reports/. Año 1995.

BASS Lenn, SOFTWARE ARCHITECTURE IN PRACTICE. 2Ed. Adisson-Wesley. ISBN 0-321-15495-9. Año 2003.

BRIAND, Devanbu, Melo, "An Investigation into coupling measures for object-oriented designs", Proceedings of the 19th International Conference on Software Engineering, ICSE '97, Boston, 412-421, 1997.

BECK, K. “Extreme Programming Explained: Embrance Change”, Pearson Education, Addison-Wesley. ISBN: 0201616416. Año 1999.

BENGTSSON, P. Design and Evaluation of Software Architecture. University of Karlskrona/Ronneby, Department of Software Engineering and Computer Science, Karlskrona. Obtenido el 13-05-2006 de: http://www.ipd.hk-r.se/pob/archive/thesis.pdf . Año 1999.

BERNARDEZ, B., DURAN, A. and GENERO, M. An Empirical Review of Use Cases Metrics for Requirements Verification. Proceedings of the SOFTWARE

MEASUREMENT EUROPEAN FORUM (SMEF'04). Rome. Accepted for publication. Año 2004.

BOEHM, Barry, PAPPACIO Philip. "Understanding and Controlling Software Costs". IEEE – Transactions on Software Engineering. Vól 14. Núm 10. Págs: 1462-1467. Año 1988.

BOEHM, Barry, RONY Ross. "Theory-W Software Project Management: Principles and Examples". IEEE – Transactions on Software Engineering. SE15. Págs: 902-916. Año 1989.

BOSCH, J., MOLIN, M., MATTSON, M., and BENGTSSON, P. Building Application Frameworks, chapter Object-oriented frameworks - Problems & Experiences. Wiley and Sons. Año 1999.

CARVALHO, A.M.B.R.; CHIOSSI, T.C.S. Introdução à Engenharia de Software. Campinas: Unicamp Editora, 2001.

CHECKLAND, Peter; HOLWELL, S. "Action Research: Its Nature and Validity , Systemic And Action Research". Vol 12 No. 1, Pag 9-22. Año 1998.

CHIDAMBER, S.R. y KEMERER, C.F. "A metric suite for object oriented design, IEEE transactions on software engineering". Vol. 20, nº6, junio, 1994, pp. 467-493.

CLEMENTS, Paul; BASS, Len; KAZMAN, Rick. "Software Architecture in Practice". Addison-Wesley. Año 2003.

DESANCTIS, Gerardine. MONGE, Peter. "Introduction to the special issue: Communication Processes for Virtual Organizations". Año 1998. http://www.ascusc.org/jcmc/vol3/issue4/desanctis.html

FAGAN, M. "Design and code Inspections to Reduce Errors in Program Development, IBM Systems Journal, vol. 15. Núm 3: Págs. 182-211. Año 1976.

FOWLER, Martin. "Is Design Dead? (Está muerto el diseño?) ". http://www.martinfowler.com/articles/designDead.html. Año 2001.

FOWLER, Martin "Refactoring: Improving the Design of Existing Code", Addison Wesley, 1999.

GALBRAITH, J. R. Designing Organizations. Jossey-Bass, San Francisco, CA. ISBN: 0-7879-5745-3. Año 1995.

GILB, Tom. “Principles of Software Engineering Management”. Wokingham England. Addison-Wesley. Año 1988.

HAIGH Peter L. ARTHUR James D., NANCE Richard E., SCHWETMAN Herbert D. A Standard Simulation Environment: A review of Preliminary Requirements. Proceedings of the Winter Simulation Conference. Año 1994.

IEEE Recommended Practice for Software Requirements Specifications (IEEE/ ANSI Standard 830– 1993). Institute of Electrical and Electronics Engineers. Año 1993.

IPL. Information Processing Ltd. Advanced Coverage Metrics for Object-Oriented Software. Año 1999.

BOOCH Grady, JACOBSON Ivar and RUMBAUGH James. The Unified Software Development Process. ISBN0-201-57169-2. Editorial Addison-Wesley. Año 2000

JONES, Capers. “Assesment and Control of Software Risks”. Englewood Cliffs. N.J. Yourdon Press. Año 1994.

KAZMAN Rick, KLEIN Mark, BARBACCI Mario, LONGSTAFF Tom, LIPSON Howard and CARRIERE Jeromy. Architecture Tradeoff Analysis. Software Engineering Institute. Carnegie Mellon University. Pittsburgh, PA 15213. kazman@sei.cmu.edu. 4th Int’l Conference on Engineering of Complex Computer Systems (ICECCS98). Año 1998 http://www.sei.cmu.edu/ata/iceccs.pdf

KAZMAN, R., CLEMENTS, P., KLEIN, M.. Evaluating Software Architectures. Methods and case studies. Editorial Addison Wesley. Año 2001.

KIM, H and BOLDYREFF, C. Developing Software Metrics Applicable to UML Models. Proceedings of the 6th International Workshop on Quantitative Approaches in Object-Oriended Software Engineering. Málaga – Spain. Pages 67-76. Año 2002.

KIRCHER Michael. JAIN Prashant. CORSARO Angelo. LEVINE David. “Distributed Extreme Programming”,XP2001, Italy, May 21-23, Año 2001. http://citeseer.ist.psu.edu/cache/papers/cs/20434/http:zSzzSzwww.cs.wustl.eduzSz~mk1zSzxp2001.pdf/kircher01distributed.pdf

LAKE. A, COOK. C, "Use of factor analysis to develop OOP software complexity metrics", Proc. 6th Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon, April 1994.

LAKHANPAL, B. “Understanding the factors Influencing the performance of software development Groups: An Exploratory Group-level Analysis”. Information and Software Technology. Num 35. Vol 8. Págs 468-473. Año 1993.

LARMAN Craig. “UML y Patrones: Una Introducción al Análisis y Diseño Orientado a Objetos y al Proceso Unificado”. Editorial Prentice Hall. 2da Edición. ISBN 84-205-3438-2. Año 2003.

LEE, Y. LIANG, B. WANG, S. "Measuring Coupling and Cohesion of an Object-Oriented Program Based On Information Flow", Proceedings of The International Conference on Software Quality (ICSQ '95), 81-90. Año 1995.

LORENZ, M, y KIDD, J. “Object oriented metrics”, Editorial Prentice-Hall, 1994

MARTIN, James. “Rapid Application Development”. New York: Macmilliam. Año 1991.

McCABE, T.J. “A Complexity Measure”. IEEE Transactions on Software Engineering, Vol. 5. Año 1976.

McCARTHY, Jim. “Dynamics of Software Development” Microsoft Press. Año 1995.

MARQUES Helena, RAMOS Rodrigo, SILVA Ismenia. “Adaptacão de um Processo de Desenvolvimento para Fábricas de Software Distribuídas”. Universidade Federal de Pernambuco. Brasil. Año 2004 http://www.spc.org.pe/ideas2004/

McCONELL Steve. “Desarrollo y Gestión de Proyectos Informáticos”. Editorial McGraw-Hill. ISBN: 1-55615-900-5. Año 1997.

Object Management Group, "OMG Unified Modeling Language Specification", Version 1.5, OMG Adopted Formal Specification formal/03-03-01, 2003.

Object Management Group, "UML 2.0 Superstructure Specification", OMG Adopted Formal Specification formal/05-07-04, 2005.

OSOY, Omar. “Organizaciones Virtuales”. http://www.monografias.com/trabajos14/organiz-virtual/organiz-virtual.shtml

PARKINSON, N. Parkinson´s Law: The Pursuit of Progress, London, Jhon Murray. Año 1958.

PFLEEGER, Shari Lawrence. Software Engineering: Theory and Practice". Editorial Prentice Hall. Año 1998.

PREECE, Jenifer. “Online Communities: Design Usability and Suporting Sociability”. Editorial WILEY. ISBN:0-471-80599-8. Año 2003

PRESSMAN Roger. Ingeniería del Software “Un Enfoque Práctico” 5 Ed. Editorial Mc Graw Hill. ISBN: 0-07-052182-4. Año 2001.

RIEL. A, "Object-Oriented Design Heuristics", Addison Wesley, 1996.

RING, P. S., VAN de Ven. “Developmental Processes of cooperative interorganizational relationships". Academy of Management Review. http://www.aom.pace.edu/amr/ . Año 1994.

Ramirez. A, P. Vanpeperstraete, A. Rueckert, K. Odutola, J. Bennett, L. Tolke, M. van der Wulp, "ArgoUML User Manual v0.16", 2004. Available from http://argouml.tigris.org.

RUP. Rational Unified Process. Version 2003.06.00.65. Copyright 1987 – 2003. Rational Software Corporation. All rights reserved.

SAEKI, M. Embedding Metrics into Information System Development Methods: An Application of Method Engineering Technique. Lecture Notes in Computer Science 2681, pp. 374– 389. Año 2003.

SCOTT Louise, JEFFERY Ross, CARVALHO Lucila, D´AMBRA Jhon . Practical Software Process Improvement – The IMPACT Project. Centre for advanced Empirical Software Engineering Research, School of Computer Science and Engineering, University of New South Wales, 2052, Sydney, Australia. WebSite http://www.caesar.unsw.edu.au/impact/index.htm. Año 2002.

SCHWETMAN Herbert D. Portable Simulation Models. Proceedings of the Winter Simulation Conference. Año 1994.

SOMERVILLE, Ian. Software engineering. Quinta Edición. Addison-Wesley, Año 1998.

TEGARDEN. D, SHEETZ. S, MONARCHI. D, "The Effectiveness of Traditional Software Metrics for Object-Oriented Systems", in: J. Nunamaker Jr, R. Sprague (eds.), Proceedings of the 25th Hawaii International Conference on Systems Sciences, Vol. IV, IEEE Computer Society Press, 359-368, Jan. 1992.

VAN GENUCHTEN, Michiel. “Why is Software Late?”. An empirical study of reasons of delay in Software Development. IEEE Transactions of Software Engineering, Vol 17. Núm 6. Págs: 582-590. Año 1991.

VASQUEZ ESCUDERO, Pedro Jesús. MORENO GARCÍA, María N. GARCÍA PEÑALVO Francisco. Métricas Orientadas a Objetos. Technical Report DPTOIA-IT-2001-002. Departamento de Informática y Automática - Universidad de Salamanca Año 2001.

WAKE, W.C. "Extreme Programming Explored". Addison-Wesley.. ISBN: 0201733978. Año 2002

YANKEE Group. "Creating and Delivering Value with Collaborative Software Development Tools". http://www.yankeegroup.com. Año 2003.

ZANONY, Roberto. Nicolas Audy, Jorge Luis. "Project Management Model: Proposal for Performance in a Physical Distributed Software Development Environment". Engineering Management Journal. http://www.asme.org/. Año 2002.

ZEIGLER Bernard P. and TUNCER Oren I. Multifacetted, Multiparadigm Modelling Perspectives: Tools for the 90´s. Proceedings of the Winter Simulation Conference. Año 1986.

Instituciones, Herramientas, Proyectos, Grupos y Personas

[1] EVOLUCION. Herramienta Software para el modelamiento y simulación en dinámica de sistemas. Grupo SIMON de investigaciones en modelos y simulación de la Universidad Industrial de Santander. – UIS - Bucaramanga. http://www.uis.edu.co/site/investigacion/grupos/simon/software/evol.html

[2] EssMODEL. (Visitado en Agosto 12 de 2006). Sitio Web de la Herramienta ElDean EssModel. [Documento WWW]. URL http://www.essmodel.com

[3] GENESIS. "Generalized Environment for Process Management In Cooperative Software Engineering". http://www.genesis-ist.org/spanish/description.htm. Septiembre 2001.

[4] GTI, Grupo de Investigación en Tecnologías de la Información – GTI perteneciente al departamento de sistemas de la Facultad de Electrónica de la Universidad del Cauca. http://gti.unicauca.edu.co

[5] HOMOS para el modelamiento de sistemas con base en objetos y reglas. Grupo SIMON de investigaciones en modelos y simulación de la Universidad Industrial de Santander – UIS - Bucaramanga. http://www.uis.edu.co/site/investigacion/grupos/simon/software/homos.html

[6] ISO. International Standards Organization. (Organización Internacional de Estándares). http://www.iso.org/iso/en/ISOOnline.frontpage

[7] MICROSOFT. Sitio Web de Microsoft Corporation. http://www.microsoft.com

[8] MOODLE. Open Source Course Management Tool. http://www.moodle.org

[9] PHPROJEKT. Open Source Groupware Suite. http://www.phprojekt.com

[10] SDMETRICS. (Visitado en Agosto 12 de 2006). Sitio Web de la Herramienta SDMetrics. [Documento WWW]. URL http://www.sdmetrics.com

[11] SEDISE (A Support Environment for Distributed Software Engineering). Grupo GTI, Línea de Interés sobre la ingeniería del software bajo ambientes de dispersión geográfica. http://groups.yahoo.com/group/sedise/

[12] SEI, Software Engineering Institute. General Information about Capability Maturity Model Integration. CMMI. http://www.sei.cmu.edu/cmmi/general/

[13] SIMEP-SW. "Proyecto para un Sistema Integral de mejoramiento de los procesos de software en Colombia". http://groups.yahoo.com/group/simep-sw/

[14] SIMON, Grupo de Investigación de modelos y simulación de la UIS. http://www.uis.edu.co/site/investigacion/grupos/simon/index.html .

[15] SPEM. Software Process Metamoldel. Metamodelo usado para describir y/o especificar visualmente los procesos de desarrollo de software. http://www.omg.org/technology/documents/formal/spem.htm

[16] SYSTEMS AND SOFTWARE CONSORTIUM."Software Productivity Consortium". http://www.software.org/pub

[17] Sitio web de la asignatura de ingeniería del software del programa de ingeniería de sistemas. Estándares de documentación para RUP. Elaborado por: Ing. Jorge Jair Moreno Chaustre. Docente Del Departamento de Sistemas. http://atenea.unicauca.edu.co/~jjmoreno/

[18] Sitio web de la Universidad del Cauca. http://www.unicauca.edu.co/

[19] Sitio web de la Universidad Industrial de Santander. http://www.uis.edu.co/

[20] UMLStudio.(Visitado 2006, Febrero 14). Herramienta CASE. [Documento WWW]. URL http://www.pragsoft.com

ANEXOS

(Espacio en Blanco Intencional)

Anexo 1. Atributos y Métricas de Calidad para la Arquitectura de Software

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 2. Arquitectura Base del ESMS-MI

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 3. Terminología de la Metodología

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 4. Artefacto “Visión de Proyecto”

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 5. Modelo de Documentación

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 6. Riesgos, Errores Clásicos y Malas Prácticas

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 7. Propuesta a COLCIENCIAS en el Marco de la Convocatoria a Centros de Excelencia.

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 8. Artículo presentado al 3er Latinoamericano y ·3er Encuentro Colombiano de Dinámica de Sistemas.

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 9. Modelo de Métricas para el Proceso

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 10. Modelo de Gestión de Riesgos

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 11. Artefacto “Plan de Proyecto”

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 12. Artefacto “Catálogo de Requisitos”

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 13. Artículo Revista “VENTANA INVESTIGATIVA”

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 14. RECURSOS DE REDACCION

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 15. ENCUESTA COLCIENCIAS-SIMEP

“Prácticas de la Ingeniería del Software en las PYMES del Sur-Occidente Colombiano”

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 16. FORMATO DE REGISTRO DE RESULTADOS DE LA EVALUACIÓN ARQUITECTÓNICA

(Espacio en Blanco Intencional para conservar el formato del documento)

Anexo 17. SUGERENCIAS A LA NUEVA ARQUITECTURA DEL ESMS-MI

(Espacio en Blanco Intencional para conservar el formato del documento)