UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Mejora de procesos en el desarrollo de software en la Coordinación de Sistemas de Cómputo (CSC) del Instituto de Ingeniería de la UNAM (IINGEN)

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Oscar Eduardo Ruíz Chávez

DIRECTOR DE TESIS

Ing. Julio Alfonso de León Razo

Ciudad Universitaria, Cd. Mx., 2017

AGRADECIMIENTOS

Con gran orgullo de haber pertenecido a la máxima casa de estudios, a ella dedico el presente trabajo de tesis donde dejo gran parte de lo aprendido en mi formación profesional.

A la Facultad de Ingeniería y a todos los profesores que durante mi carrera, además de sus enseñanzas, con gran cariño compartieron sus experiencias personales y profesionales.

Al Instituto de Ingeniería por haberme permitido participar en este proyecto de mejora, brindándome las herramientas necesarias para llevarlo a cabo.

Al Ing. Marco Ambriz Maguey por darme la oportunidad de integrarme a la Coordinación de Sistemas de Cómputo donde he trabajado de la mano con personas agradables, dedicadas y muy profesionales.

A mi director de tesis Ing. Julio Alfonso de León por todo el apoyo brindado a lo largo de mi permanencia en el Instituto de Ingeniería, por la confianza, por su tiempo y sus enseñanzas.

A mis padres, Maria Guadalupe y Victor Sergio de todo corazón por darme la vida, por hacerme una persona de bien, por darme la oportunidad de estudiar con grandes esfuerzos, por apoyarme en todo momento, por aconsejarme y por estar siempre conmigo en las buenas y en las malas.

A mis hermanos Ericka Teresa y Victor Sergio por acompañarme y ayudarme siempre, brindándome su cariño y contagiándome del entusiasmo y la fuerza para salir adelante. A ellos también agradezco la bendición que han traído a la familia con la llegada de mis sobrinos Sergio Said, Aranza Saraí e Iriel Sánchez, un motivo más para esforzarnos y ser mejores cada día.

A mis amigos Jessica Ibarra, Fernando Mejía, Eduardo Guevara y Ricardo Montano por su apoyo incondicional y por ser los cómplices de todos mis logros y lo bueno que me ha pasado.

A Nora Vanesa por quererme tanto y apoyarme siempre, por ser una mujer inteligente, emprendedora y un ejemplo a seguir.

A todos mis familiares que a lo largo de mi carrera me ayudaron, especialmente a quienes ya no están con nosotros físicamente pero siguen presentes en nuestros corazones.

Hago una dedicatoria especial de este trabajo a mi tía Irma Antonia Ruiz Ibañez (†) por creer en mí, apoyarme y quererme tanto, sé que has estado y estarás conmigo en todos mis logros.

ÍNDICE

Capítulo 1

Introducción

Actualmente, debido a las necesidades de comunicación y movilidad, el desarrollo de software ha tenido que evolucionar hacia un enfoque de Ingeniería. La presencia de nuevas tecnologías y avances en las redes de datos, nos ha permitido tener a la mano grandes cantidades de información y nos han facilitado nuestras tareas cotidianas, a tal punto de crear una dependencia cada vez mayor. Si bien es cierto que el software está implícito en todo lo que hacemos, también es cierto que existe una gran crisis al construirlo, ya que al ser un producto intangible no se percibe o dimensiona tan fácilmente el proceso que hay detrás. Debido a lo anterior son constantes las fallas presentes en el mismo y se encuentran aquí algunas áreas de oportunidad sobre todo en su proceso de elaboración. Esta es una de las razones por las que la Ingeniería de Software toma una mayor importancia al proporcionar los pasos a seguir para crear software de calidad.

La Coordinación de Sistemas de Cómputo del Instituto de Ingeniería de la UNAM (CSC IINGEN UNAM), no se encuentra al margen de la crisis mencionada anteriormente, por lo que es considerado como tema de gran interés el proponer un proceso de mejora para el desarrollo de software dentro de la misma. Este trabajo de tesis tratará los siguientes objetivos para sentar las bases:

- Marco teórico: Conocer a grandes rasgos aquellos modelos, metodologías, estándares y herramientas de la Ingeniería de Software, con la intención de tener una referencia básica de éstos y que sea accesible a los miembros del grupo de trabajo del IINGEN.

- Mejora de procesos: Proponer una metodología que permita realizar software de una manera más estructurada para así lograr mayores niveles de madurez en su elaboración, tomando como base el proceso general del ciclo de vida del software (toma de requerimientos, análisis, diseño y construcción).

- Definir y estandarizar documentos: Contar con documentación de utilidad para fortalecer la base de conocimiento de los sistemas de software que se desarrollan en el IINGEN facilitando el mantenimiento, actualización y migración cuando se requiera.

Para satisfacer dichos objetivos, se ha contemplado la siguiente estructura, la cual permitirá abordar cada uno de ellos:

Antecedentes

Este capítulo nos sitúa en el contexto histórico y actual del Instituto de Ingeniería de la UNAM, resaltando la importancia de los proyectos de investigación realizados dentro del mismo. Se mencionan también algunos de los sistemas de información desarrollados en el IINGEN y plantea el compromiso a futuro de preservar su papel como principal actor del desarrollo tecnológico en nuestro país.

Tópicos generales

En este capítulo se presenta el marco teórico para sentar las bases de cada uno de los objetivos planteados. Se abarcan conceptos importantes dentro de la Ingeniería de Software y se contemplan algunos estándares, metodologías y herramientas útiles.

Identificar áreas de oportunidad para la mejora de procesos

En este capítulo, mediante un análisis de la situación actual del IINGEN, se definen las áreas de oportunidad que tendrían un mayor impacto en las prácticas de Ingeniería de Software en la primera etapa de mejora de procesos.

Plan de mejora de procesos

En este capítulo se hace la propuesta de una metodología iterativa de 5 pasos la cual se empleará para lograr aumentar el nivel de madurez en el desarrollo de software.

Estandarizar las herramientas a emplear en la mejora de procesos.

En este capítulo se define la documentación de valor, siendo la mínima necesaria para cada proyecto realizado en el IINGEN. Se establecen los documentos necesarios para llevar a cabo una buena toma de requerimientos de acuerdo al tipo de proyecto, se normaliza el estilo de código que utilizarán los desarrolladores del IINGEN y se especifica el repositorio de código a emplearse.

Revisar plan de mejora en etapa uno

En este capítulo se hace una comparación grafica de acuerdo a los scripts de evaluación, utilizados para revisar los aspectos que se lograron una vez implementada la metodología de los 5 pasos.

Capítulo 2

Antecedentes

2.1 El Instituto de Ingeniería

El Instituto de Ingeniería de la Universidad Nacional Autónoma de México (IINGEN UNAM) es el centro de investigación más productivo del país, pues su misión ha sido contribuir al desarrollo de éste y al bienestar de la sociedad a través de la investigación en diversas áreas de la Ingeniería, la formación de recursos humanos y la vinculación con la sociedad.

La comunidad del Instituto de Ingeniería está integrada por investigadores, técnicos académicos, personal administrativo y una amplia población de becarios que trabajan en la realización de tesis de licenciatura, maestría y doctorado.

Desde su fundación en el año de 1956, el Instituto de Ingeniería realiza proyectos de investigación enfocados a problemas generales de la Ingeniería, colaborando con entidades tanto públicas como privadas. Asimismo, proporciona servicios de Ingeniería a los diversos sectores de la sociedad y ha puesto énfasis en la formación de recursos humanos y en difundir los resultados de sus investigaciones contribuyendo así al desarrollo del país y el bienestar de la sociedad.

Algunos de los proyectos son financiados con recursos que la UNAM otorga, y la mayor parte, mediante contratos de investigación con empresas solicitantes. Por ello, el prestigio del Instituto de Ingeniería es ampliamente reconocido.

En la actualidad, las áreas que cubre el Instituto incluyen una rica mezcla de las disciplinas e interdisciplinas de la Ingeniería moderna. Los temas de tesis que se dirigen dentro del Instituto y los artículos que se publican representan una muestra muy variada de lo mejor de la Ingeniería nacional, que honra la prestigiada tradición del Instituto de Ingeniería de la UNAM.

Entre las especialidades que se cultivan actualmente dentro del IINGEN se encuentran la Ingeniería Estructural, que estudia entre otras cosas el comportamiento estructural de edificios y el comportamiento dinámico de los materiales; la Ingeniería en Riesgos Naturales cuya finalidad es contar con

evaluaciones probabilistas de riesgo ante amenazas naturales como son sismos, huracanes, inundaciones y granizo; la Ingeniería Hidráulica, que estudia el control de inundaciones, estructuras hidráulicas entre otras cosas; la Ingeniería Eléctrica Electrónica que abarca las áreas de Control Automático, Sistemas Eléctricos de Potencia, Autómatas Celulares, la Ingeniería Lingüística y la Optoelectrónica; la Ingeniería Mecánica cuyas líneas de investigación se enfocan al diseño y desarrollo de todo tipo de máquinas y mecanismos; la Ingeniería en Computación, cuyas ramas de estudio abarcan al diseño y manejo de bases de datos, las redes de datos y el desarrollo de software; la Ingeniería en Telecomunicaciones que abarca desde la simulación de amplificadores ultra-rápidos, hasta la fabricación de satélites de comunicación educacionales para entrenar y atraer estudiantes al mundo de la ciencia y la tecnología de punta.

En el futuro, el Instituto prevé preservar su papel de árbitro nacional de la Ingeniería y actor principal del desarrollo tecnológico. Al mismo tiempo, apoyará de manera más efectiva a la docencia y la formación de expertos, conjuntamente con la Facultad de Ingeniería, como siempre, y acrecentará su participación en programas universitarios de punta, así como la vinculación con la industria, el desarrollo de nuevas tecnologías y la colaboración con instituciones afines.

2.2 Sistemas de información desarrollados en el IINGEN

En el Instituto de Ingeniería se han desarrollado varios sistemas de información con el fin de automatizar procesos en las diferentes áreas del Instituto, facilitando así la realización de sus diversas tareas. A continuación se listan algunos de los más importantes:

SISTEMA DE CONTROL DE ESTUDIANTES (SICOE)

El sistema de control de estudiantes SICOE, es un sistema web desarrollado para el área académica del Instituto de Ingeniería. Este sistema permite llevar un control digital de la información de los estudiantes que realizan actividades en el Instituto con los diferentes académicos, ya sea en servicio social, tesis de licenciatura, maestría o doctorado.

SISTEMA DE CONSULTA DE PROYECTOS

Consulta de proyectos es un sistema web y móvil creado para el área administrativa del Instituto de Ingeniería. El fin de este sistema es mantener al tanto a los Investigadores de la situación financiera de los proyectos que tienen a su cargo.

SISTEMA DE BASE DE DATOS ACADÉMICA DEL INSTITUTO DE INGENIERÍA (SBDAII)

Es un sistema para la administración de la información académica de los investigadores y técnicos académicos del Instituto de Ingeniería. Gran parte de la información de la vida académica del personal del Instituto de Ingeniería se encuentra almacenada en este sistema gracias a la información proporcionada por los mismos investigadores y técnicos.

COBRANZA DE CONVENIOS

Es un sistema web creado para el área administrativa del Instituto de Ingeniería. En este sistema se lleva a cabo el control de pagos de convenios que se realizan en el Instituto de Ingeniería con patrocinios externos.

Existen otros sistemas desarrollados en el Instituto de Ingeniería como es el caso de shakemaps, sismos fuertes, entre otros.

2.3 Situación actual

Actualmente el Instituto de Ingeniería cuenta con varios sistemas de información desarrollados, sin embargo, la escasa documentación de los mismos, las diversas maneras de codificar utilizada por cada uno de los desarrolladores e inclusive el no contar con un repositorio de respaldo y de manejo de versiones, propicia que las actualizaciones y mantenimiento de los mismos sea demasiado tortuoso y demandante en cuanto al uso de recursos principalmente de desarrolladores.

Adicional a lo anterior, tampoco existe una práctica sana para llevar acabo la toma de requerimientos, tanto de nuevos sistemas como de los ya existentes que requieren de actualizaciones en cuanto a funcionalidad. Esta es la principal causa de que en ocasiones los desarrollos no se ajusten en su totalidad a los procesos requeridos con sus consecuentes modificaciones o se lleguen a presentar problemas en el alcance de los mismos.

2.4 Referencias:

Referencias web:

[1] Sitio web del Instituto de Ingeniería
http://www.iingen.unam.mx
Noviembre 2014

[2] Secretaria Académica Informe 2006 SICOE
http://eventos.iingen.unam.mx/Informe2005_2006/C_11_4.html

[3] Secretaria Académica Informe 2006 SBDAII
http://eventos.iingen.unam.mx/Informe2005_2006/C_11_3.html

[4] Sistema de consulta de proyectos del SIRF
http://www.iingen.unam.mx/es-mx/Investigacion/Proyectos/Lists/Proyectos/Attachments/200/MarcoAmbriz.pdf

Capítulo 3

Tópicos generales

3.1 Software

Según el estándar 610.12-1990 de la IEEE el software es un conjunto de programas, procesamientos, rutinas y posible documentación asociada con la operación de un sistema de cómputo.

El software es utilizado para dispositivos electrónicos de cualquier tamaño y arquitectura. Sin el software, un dispositivo electrónico no es más que un objeto sin utilidad.

3.1.1 Clasificación del software

El software se clasifica principalmente en dos categorías:

1.- Software base: En esta clasificación se encuentra el software de sistema o software operativo (sistemas operativos). El software operativo procesa tareas esenciales como el control y la gestión eficaz de todos los recursos de hardware.

2.-Software aplicativo: Es todo aquel software cuyo propósito es ayudar al usuario a realizar una tarea. El software de aplicación se puede considerar como una herramienta que extiende las capacidades humanas, permitiendo la realización de tareas que de otro modo sería difícil o imposible realizarlas. Por lo tanto, la mayor parte del software cae dentro de esta clasificación. El software de aplicación lleva a cabo tareas tales como el tratamiento de textos, gestión de bases de datos y similares.

El software aplicativo a su vez se clasifica en:
a) Software a la medida: Es el software desarrollado para un cliente en particular.
b) Software de propósito general: Es el software desarrollado para el mercado en general.

3.1.2 Importancia del software

La importancia del software reside principalmente en que hace posible la distribución del producto más importante de nuestro tiempo: **La información**. La información es pieza fundamental en la toma de decisiones de cualquier lugar ya que a mayor calidad de información, mejor es la toma de decisiones.
Ya sea que resida en un teléfono móvil u opere en el interior de una computadora central, el software es un transformador de información. El software produce, administra, adquiere, modifica, despliega o transmite información que puede ser tan simple como un solo bit o tan compleja como una presentación con multimedios generada a partir de datos obtenidos de decenas de fuentes independientes.

3.1.3 Participantes en el desarrollo del software

El desarrollo de software es una actividad que, dada su complejidad, debe realizarse en grupo. Esta actividad requiere de distintas capacidades, imposible de encontrarlas todas en una sola persona. Por ello, se hace necesario formar un **grupo de desarrollo** con las personas que cubran todas las capacidades requeridas para llevar a cabo con éxito el desarrollo.
Cada persona debe tener un rol dentro del grupo, que viene dado por su experiencia y capacidades personales. Los roles que tradicionalmente se consideran en el desarrollo de software son: administrador de proyecto, analista, diseñador, programador, téster, asegurador de calidad, documentador, ingeniero de manutención, ingeniero de validación y verificación, administrador de la configuración. Para cada uno de estos roles, se definen sus objetivos, actividades, interacción con otros roles, herramientas a utilizar, perfil de las personas en ese rol y un plan de trabajo.

Es importante tomar en cuenta lo siguiente:

- Es posible que no se requieran todos los roles en un desarrollo. Eso dependerá del tamaño y del tipo del desarrollo.
- Habrá ocasiones en que se requerirán algunos roles en menor medida y otros en mayor.
- Es posible también que una persona realice las labores de más de un rol al mismo tiempo. Esto, sobre todo en proyectos de desarrollo de software más pequeños.
- Es posible tener más de una persona con un mismo rol en un equipo de desarrollo.

El hecho de que en un grupo de desarrollo no se tengan claro los roles, las responsabilidades y actividades asociadas, hace que se produzcan problemas.

Se generan problemas cuando:

- Una o más actividades no están asociadas a ningún rol.
- Una o más actividades están asociadas a más de un rol.

Por lo anterior, se hace necesario que cada miembro conozca muy bien su rol dentro del proyecto, así como las responsabilidades y actividades asignadas.

Cabe mencionar que durante el desarrollo del software, es necesario que el grupo de desarrollo esté en constante comunicación con dos personajes importantes: el cliente y el usuario. Ver **figura 1**.

- El cliente: Es la compañía, organización o persona que está pagando por el sistema que se va a desarrollar.
- El usuario: Es la persona o personas que realmente usarán el sistema; son quienes se sientan frente a la computadora, ingresan los datos y leen las salidas.

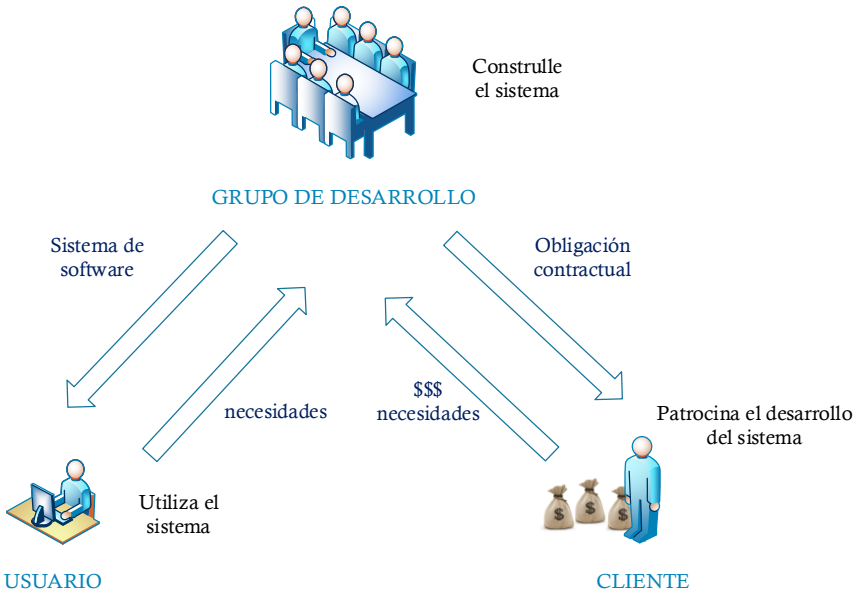


Figura 1. Participantes en el desarrollo del software

3.1.4 El software en la actualidad

Hoy en día debido a las necesidades de comunicación y movilidad, surgieron dos conceptos que trajeron grandes cambios en la forma de desarrollar y distribuir el software:

Computación ubicua: Es la integración de dispositivos en la vida cotidiana de las personas con los cuales pueden interactuar de manera inconsciente para realizar sus actividades diarias. Un ejemplo sencillo pero ilustrativo de esto son los smartphones que incorporan aplicaciones que utilizamos frecuentemente como el blog de notas, la agenda telefónica, entre muchas otras que permiten la transmisión de información mediante redes de corto alcance; Un ejemplo más son los aparatos electrodomésticos que cada vez son más completos en funcionalidades y que con la domótica pueden comunicarse e interactuar entre ellos. Por ejemplo: En un hogar el refrigerador, la estufa y el horno pueden programarse para que se comuniquen e interactúen entre ellos. En el momento en que el usuario indica de alguna forma lo que desea cocinar, la nevera por medio de sensores puede revisar los ingredientes

que hay y los que hacen falta para la receta y la estufa indicar el tiempo de cocción del alimento (si el refrigerador detecta que algún alimento necesita ser descongelado, la información se transmite a la estufa o al horno para que se lleve a cabo dicha acción). La computación ubicua no implica el uso del internet en los dispositivos para llevar a cabo sus funciones, en el siguiente concepto si se incorpora esta idea.

Internet de las cosas o Internet of Things (IoT): Este modelo de internet contempla una interacción basada en la computación ubicua. Se trata de “un nuevo internet” que permite la interacción entre objetos y personas en cualquier lugar y momento mediante dispositivos fácilmente integrables en hogares, medios de trabajo y lugares públicos, para procesar y transmitir información mediante internet, proporcionando servicios y aplicaciones inteligentes. Un buen ejemplo de ello son los smartwach que tienen la posibilidad de censar diferentes aspectos del usuario como puede ser los kilómetros recorridos y el número de calorías utilizados al correr o caminar. Los datos recopilados se sincronizan con un smartphone y mediante este pueden ser transmitidos por la red a otros dispositivos o sistemas de almacenamiento.

Los conceptos anteriores, obligaron al desarrollo y distribución del software de una forma por completo diferente a la acostumbrada (aplicaciones de escritorio) mediante las llamadas aplicaciones web y móviles:

- La mayoría del software se distribuye por internet ya sea que se descargue directo al dispositivo en el caso de las aplicaciones móviles o se consuma directamente desde la web en el caso de las aplicaciones web.
- En las aplicaciones web, se dispone siempre de una versión actualizada, sin involucrar al usuario en tareas de actualización y en el caso de las aplicaciones móviles el proceso es automático o se notifica cuando existen nuevas versiones para que el usuario de manera sencilla lleve a cabo la actualización.

- Las aplicaciones web son accesibles y operables desde cualquier plataforma y ubicación (al no ser necesario su descarga para una posterior instalación o configuración, son capaces de distribuirse con mayor facilidad a un elevado número de usuarios).
- Cuando se hace un cambio o corrección en las aplicaciones web, se actualiza la funcionalidad y contenido a cada usuario, todo de forma instantánea. Esto es una ventaja y a la vez es una desventaja ya que si al llevar a cabo un cambio en el software cometemos un error, todos los usuarios lo notarán en sus equipos instantáneamente. Con esta última característica podemos ver la importancia de poner en práctica una buena Ingeniería de Software.

¿Cuál es la mejor manera de hacer software?

Para triunfar en la creación de software (satisfacer las necesidades de las personas que lo usan, trabajar sin fallos durante largos periodos, que sea fácil de modificar e incluso más fácil de usar), se necesita disciplina al momento de diseñarlo y construirlo. Esta es la razón por la que **es necesario un enfoque de Ingeniería en la creación del software**.

Atributos de un buen software

- Mantenibilidad: El software debe poder evolucionar para cumplir con las necesidades de cambio de los clientes.
- Confiabilidad: El software debe ser fiable, seguro, no debe causar daños físicos o económicos en el caso de una falla del sistema.
- Eficiencia: El software debe aprovechar al máximo los recursos del sistema.
- Usabilidad: El software debe ser fácil de utilizar.

3.2 Ingeniería de Software

Según el estándar 610.12-1990 de la IEEE la Ingeniería de Software es:
1) La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la Ingeniería al software.
2) El estudio de enfoques según el punto 1.

La Ingeniería de Software está formada por varias capas: estándares, metodologías y un conjunto de herramientas que permite a los profesionales elaborar software de alta calidad. Ver **figura 2**.

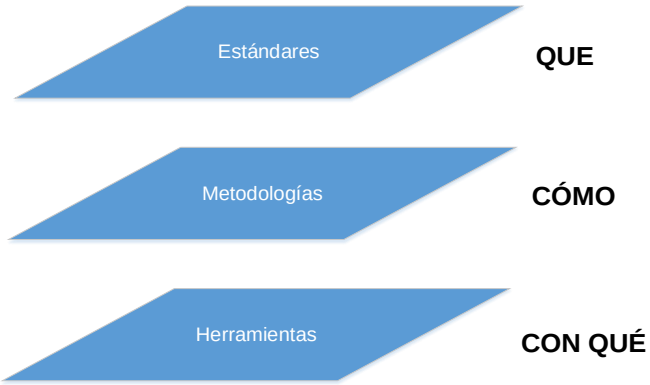


Figura 2. Capas de la Ingeniería de Software

3.2.1 Modelos de proceso

Proceso de desarrollo de software: Un proceso de desarrollo del software consta de un conjunto de actividades, acciones y tareas que conducen a la creación de un producto de software.

- Actividad: Una actividad busca lograr un objetivo amplio y se desarrolla sin importar el dominio de la aplicación, tamaño del proyecto, complejidad del esfuerzo o grado de rigor con el que se usará la Ingeniería de Software.
- Acción: Una acción es un conjunto de tareas que producen un producto importante del trabajo.
- Tarea: Una tarea se centra en un objetivo pequeño pero bien definido que produce un resultado tangible.

Cada una de las actividades, acciones y tareas se encuentra dentro de una estructura o modelo que define su relación tanto con el proceso como entre sí.

Una estructura de proceso general para la Ingeniería de Software consta de cinco actividades estructurales: comunicación, planeación, modelado, construcción y despliegue. Además, a lo largo de todo el proceso se aplica un conjunto de actividades sombrilla: seguimiento y control del proyecto, administración de riesgos, aseguramiento de la calidad, administración de la configuración, revisiones técnicas, entre otras. Ver **figura 3**.

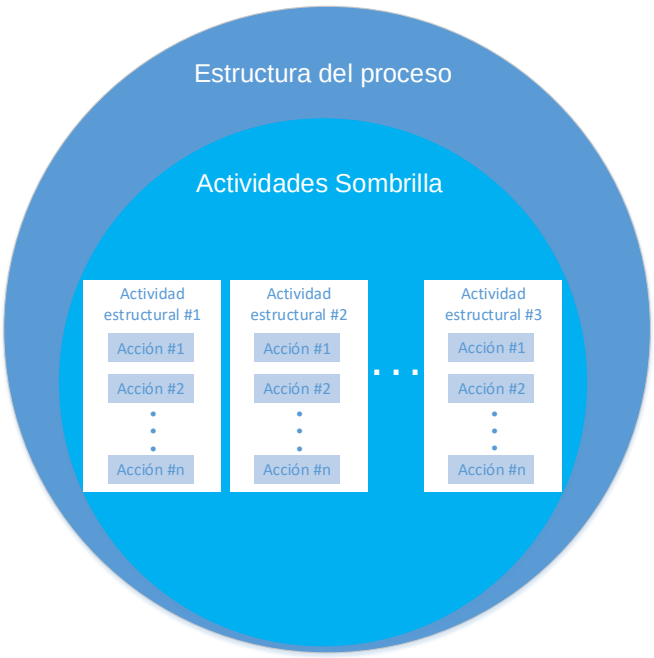


Figura 3. Estructura o modelo de proceso para la Ingeniería de Software

Todos los modelos de proceso del software pueden incluir las actividades estructurales generales pero cada una pone distinto énfasis en ellas y define en forma diferente el flujo de proceso que invoca cada actividad estructural.

3.2.1.1 Modelo de la cascada

El modelo de la cascada o ciclo de vida en cascada nace con el surgimiento de la Ingeniería de Software propiamente como un área. Por sus características este modelo es aplicable a proyectos pequeños donde desde un inicio los requisitos están bien entendidos.

El modelo de la cascada se divide en etapas claramente especificadas, conservando siempre la misma idea, de que no puede empezar una etapa hasta que no se termine la anterior. Cada etapa corresponde a una actividad de desarrollo efectuada por un grupo de personas especializadas en esa tarea y que generan un conjunto de documentos como cierre de la etapa. Este modelo se enfoca a la producción de documentos. Ver **figura 4**.

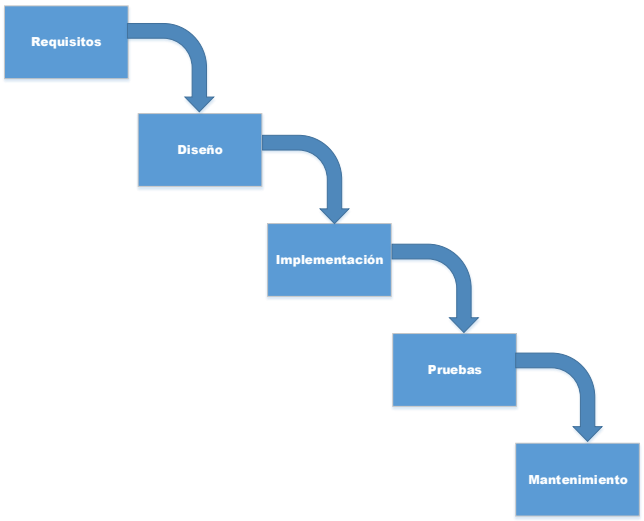


Figura 4. Modelo de la cascada

3.2.1.2 Modelo iterativo

En un modelo iterativo se mejoran los requisitos (más genéricos a más específicos). En cada ciclo de iteración se revisa y mejora el producto. Es importante señalar que este modelo no implica añadir funcionalidades en el producto solo revisión y mejora. Ver **figura 5**.

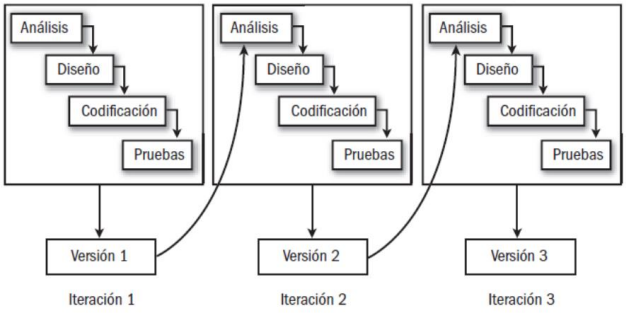


Figura 5. Modelo iterativo

3.2.1.3 Modelo incremental

En un modelo incremental se desarrolla por partes el producto de software, integrando funcionalidades al sistema a medida que se completan las mismas. Este modelo aplica en forma iterativa elementos del modelo en cascada. Se entrega un producto operacional completamente funcional en cada incremento. Cada nuevo incremento se realiza sobre el anterior. Ver **figura 6**.

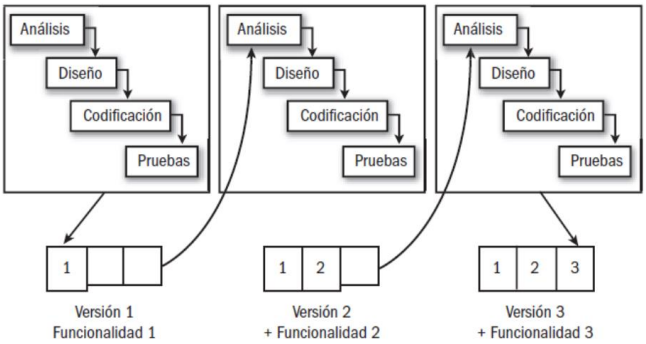


Figura 6. Modelo incremental

3.2.1.4 Modelo de proceso evolutivo

El software, como todos los sistemas complejos, evoluciona en el tiempo. Es frecuente que los requerimientos del negocio y del producto cambien conforme avanza el desarrollo, lo que hace que no sea realista trazar una trayectoria rectilínea hacia el producto final. Se necesita un modelo de proceso diseñado explícitamente para adaptarse a un producto que evoluciona con el tiempo.

Los modelos evolutivos se caracterizan por la manera en la que permiten desarrollar versiones cada vez más completas del software. A continuación se presentan dos modelos comunes de proceso evolutivo:

- Prototipos.
- Espiral.

3.2.1.4.1 Prototipos

Un prototipo es una versión preliminar, intencionalmente incompleta y en menor escala de un sistema. El paradigma de hacer prototipos ayuda a mejorar la comprensión de lo que hay que elaborar cuando los requerimientos no están claros.

El paradigma de construcción de prototipos comienza con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos generales para el software, identifican los requisitos conocidos y las áreas en las que es imprescindible una mayor definición. Entonces aparece un diseño rápido. El diseño rápido se centra en una representación de los aspectos del software que serán visibles para el usuario/cliente. El diseño rápido lleva a la construcción de un prototipo. El prototipo lo evalúa el cliente/usuario y se utiliza para refinar los requisitos del software a desarrollar. La iteración ocurre cuando el prototipo se pone a punto para satisfacer las necesidades del cliente, permitiendo al mismo tiempo que el desarrollador comprenda mejor lo que se necesita hacer. Ver **figura 7**.

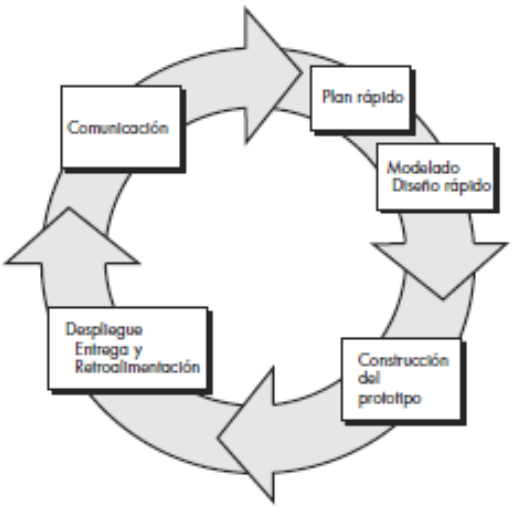


Figura 7. Modelo de hacer prototipos

Para la construcción de un prototipo, pueden utilizarse fragmentos de programas existentes o aplicar herramientas que permitan generar rápidamente programas que funcionen.

3.2.1.4.2 Espiral

El modelo en espiral está pensado para proyectos largos, caros y complicados como puede ser, por ejemplo, la creación de un sistema operativo. Combina las características del modelo de prototipos y el modelo en cascada.

El modelo en espiral se acopla con la naturaleza iterativa de hacer prototipos permitiendo desarrollar versiones más completas del software en cada iteración. Cada versión construida se conoce como prototipo. Antes de generar un nuevo prototipo, se evalúa el riesgo que se corre si continuáramos con la siguiente iteración, es decir, si aún por parte del cliente y de nosotros existe tiempo, recursos y decisión para seguir mejorando este proceso de desarrollo o por el contrario, si encontramos que los riesgos son demasiado grandes, pues entonces tomar las decisiones adecuadas como la suspensión del proyecto o la solicitud de más

recursos. La idea es que antes de generar un prototipo, evaluemos la factibilidad y analicemos los riesgos. Si el resultado es positivo, entonces generamos el primer prototipo (el cual abarca sólo una parte del sistema), continuamos con las fases siguientes (simulación, comparación, validación de requerimientos) hasta que deje de ser prototipo y se convierta en el sistema completo, cuya puesta en operación, ya integrada y aceptada por el cliente, constituye la salida del proceso. Ver **figura 8**.

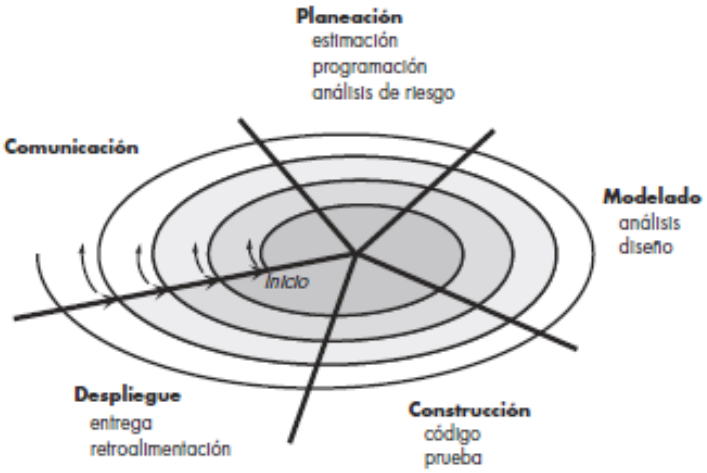


Figura 8. Modelo de espiral

3.2.2 Metodologías

Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo indicando:

- Qué personas deben participar en el desarrollo de las actividades y qué papel deben tener.
- Pasos a seguir y cómo realizarlos para finalizar una tarea.
- La información que se debe producir como resultado de una actividad y la información necesaria para comenzarla.

Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, iterativo, incremental…) y suelen clasificarse en dos grupos:

- Metodologías tradicionales.
- Metodologías ágiles.

3.2.2.1 Metodologías tradicionales

Las metodologías tradicionales se basan en una fuerte planificación y en el uso exhaustivo de documentación durante todo el ciclo del proyecto.

3.2.2.1.1 Rational Unified Process (RUP)

Es una metodología de desarrollo basada en UML iterativa e incremental enfocada hacia los casos de uso, la arquitectura y el manejo de riesgos. RUP fue desarrollado por Rational Software a mediados de los 90's. En el 2002, IBM adquiere Rational Software y RUP es avalado como un estándar en el ámbito internacional.

El objetivo de RUP es asegurar la producción de software de alta calidad que satisfaga la necesidad del usuario final dentro de un tiempo y presupuesto previsible. El proceso de RUP debe adaptarse a las características de la organización para la que se está desarrollando el software.

RUP mejora la productividad del equipo ya que permite que cada miembro del grupo sin importar su responsabilidad específica acceda a la misma base de datos de conocimiento. Esto hace que todos compartan el mismo lenguaje, la misma visión y el mismo proceso acerca de cómo desarrollar software.

Las seis mejores prácticas de desarrollo que aplica RUP son las siguientes:

- Gestión de requerimientos.
- Desarrollo de software en forma iterativa.
- Uso de arquitecturas basadas en componentes.
- Modelado visual del software.
- Verificación de la calidad del software.
- Control de cambios.

El ciclo de vida RUP es una implementación del modelo en espiral. Se establecen tareas en fases e iteraciones. Gráficamente, el ciclo de vida de RUP puede representarse en dos dimensiones. Ver **figura 9**.

- El eje horizontal representa el tiempo y muestra el aspecto dinámico del proceso, expresado en término de ciclos, fases, iteraciones y metas.
- El eje vertical representa el aspecto estático del proceso y se expresa en términos de actividades, artefactos, trabajadores y flujos de trabajo.

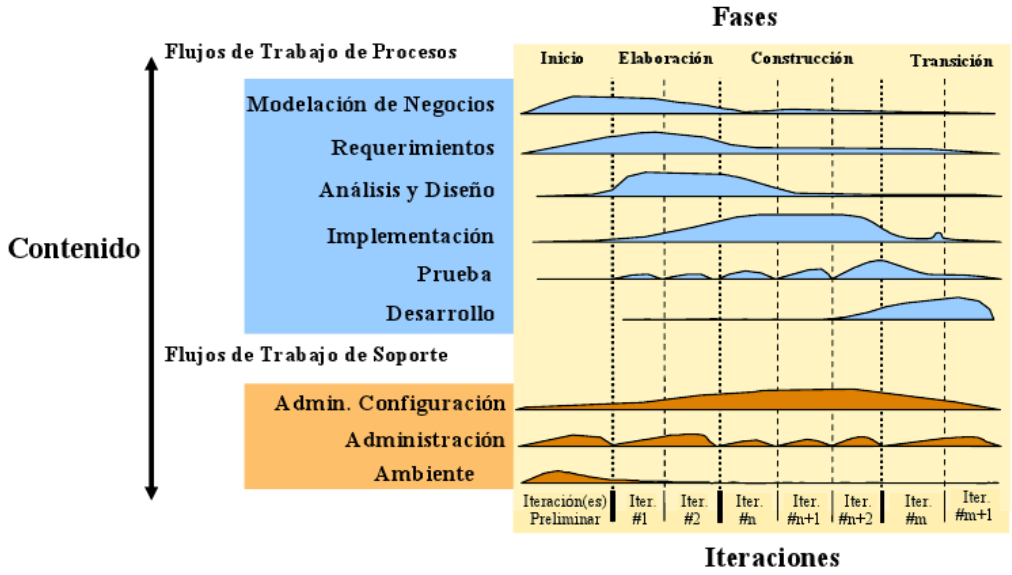


Figura 9. Ciclo de vida de RUP

Elementos estáticos de RUP

- Actividades: Son los procesos que se llegan a determinar en cada iteración.
- Trabajadores: Son las personas o entes involucrados en cada proceso.
- Artefactos: Un artefacto puede ser un documento, un modelo, o un elemento de un modelo.

Elementos dinámicos de RUP

Las fases del proceso:

- Inicio (define el alcance del proyecto): Consiste en evaluar el proyecto, se obtiene una visión general de los requerimientos del proyecto y se decide llevar adelante o no el proyecto en función de los recursos requeridos, se determinan los principales casos de uso y se hace un primer esbozo de arquitectura.
- Elaboración (definición, análisis, diseño): Esta fase tiene como objetivo construir la arquitectura del sistema. Una vez concluida la elaboración, se conocen definitivamente las exigencias del proyecto y su arquitectura. También se tiene una lista de riesgos.
- Construcción (implementación): Corresponde al desarrollo de software de la arquitectura de manera incremental.
- Transición (fin del proyecto y puesta en producción): Comprende la instalación del software en los equipos del cliente.

Disciplinas a realizar en cada fase del proyecto

RUP define nueve disciplinas a realizar en cada fase del proyecto:

1. Disciplinas de desarrollo:
    - Modelado empresarial.
    - Requisitos.
    - Análisis y diseño.
    - Implementación.
    - Prueba.
    - Despliegue.

2. Disciplina de soporte
    - Gestión de cambios y configuración.
    - Gestión de proyectos.
    - Entorno.

Cada una de estas disciplinas es desarrollada mediante un ciclo de iteraciones. Es recomendable que a cada una de estas iteraciones se les clasifique y ordene según su prioridad, y que cada una pueda ser convertida luego, en un entregable al cliente. Esto trae como beneficio una conveniente y necesaria retroalimentación que se tendría en cada paso que signifique un entregable o en cada iteración.

RUP es una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo de software.

3.2.2.2 Metodologías ágiles

Las metodologías ágiles enfatizan en la “agilidad” del proyecto y siguen un conjunto de principios que conducen a un enfoque más informal (pero no menos efectivo) del proceso de software. Las metodologías ágiles tienen como base los 12 principios del manifiesto para el desarrollo ágil de software. Ver **apéndice**.

3.2.2.2.1 Xtreme Programming (XP)

Xtreme Programming es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes y simplicidad en las soluciones implementadas. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

ELEMENTOS DE LA METODOLOGÍA XP

Historias de usuario: Es una técnica utilizada para especificar formalmente los requisitos funcionales y operativos de un producto de software a ser desarrollado. Se maneja con una serie de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales.

- El tratamiento de las historias de usuario debe ser muy dinámico y flexible, queriendo decir con esto que los contenidos deben permanentemente ser actualizados y ser empleados dependiendo del avance del trabajo.

- Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan hacerla e implementarla en unas pocas semanas.
- Las historias de usuario se descomponen en tareas de programación y se asignan a los programadores para ser implementadas durante una iteración.

Roles en la metodología XP:

- Programador: Escribe las pruebas unitarias y produce el código del sistema.
- Cliente: Escribe las historias de usuario y las pruebas funcionales para validar su Implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.
- Encargado de pruebas (Tester): Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- Encargado del seguimiento (Tracker): Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- Entrenador (Coach): Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- Consultor: Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.
- Gestor (Big Boss): Es el elemento humano que vincula los clientes y los usuarios con los programadores, colaborando a fin de que el equipo trabaje efectivamente y creando las condiciones adecuadas. Su labor esencial es de coordinación.

Ciclo de vida ideal de XP:

1. Exploración.
2. Planificación de la entrega.
3. Iteraciones.
4. Producción.
5. Mantenimiento.
6. Cierre del proyecto.

Proceso XP: El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El programador construye ese valor.
4. Vuelve al paso 1.

Actividades: XP describe cuatro actividades que se llevan a cabo dentro del proceso de desarrollo de software.

1.- Codificar: El código, dicen los partidarios de esta posición, es siempre claro y conciso y no se puede interpretar de más de una forma.

- Si nos enfrentamos con varias alternativas para un problema de programación, uno debiera simplemente codificar todas las soluciones y determinar con pruebas automatizadas qué solución es la más adecuada.
- Un programador que trate con un problema de programación complejo y encuentre difícil explicar la solución al resto, podría codificarlo y usar el código para demostrar lo que quería decir. Otros programadores pueden dar retroalimentación de ese código codificando también sus pensamientos.

2.- Probar: Se llevan a cabo dos tipos de pruebas:

- Pruebas unitarias: Son pruebas automatizadas que prueban el código.
- Pruebas de aceptación: Basadas en los requisitos dados por el cliente.

3.- Escuchar: Para que los programadores encuentren cual debe ser la funcionalidad del sistema, deben escuchar. Tienen que escuchar las necesidades de los clientes. También tienen que intentar entender el problema del negocio y dar a los clientes retroalimentación sobre el problema, para mejorar el propio entendimiento del cliente sobre el problema.

4.- Diseñar: Buenos diseños evitarán pérdidas de dependencias dentro de un sistema; esto significa que cambiar una parte del sistema no tendrá por qué afectar a otras.

Prácticas en la metodología Xtreme Programming:

- El juego de la planificación: Hay una comunicación frecuente entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.
- Entregas pequeñas: Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más de 3 meses.
- Metáfora: El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema.
- Diseño simple: Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

- Pruebas: La producción de código está dirigida por las pruebas unitarias. Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.
- Refactorización (refactoring): Es una actividad constante de reestructuración del código con el objetivo de evitar duplicación del código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.
- Programación en parejas: Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores…).
- Propiedad colectiva del código: Cualquier programador puede cambiar cualquier parte del código en cualquier momento.
- Integración continua: Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
- 40 horas por semana: Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.
- Cliente in-situ: El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor al negocio y los programadores pueden resolver de manera más inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.
- Estándares de programación: XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

Principios básicos de la programación extrema:

Simplicidad: La simplicidad es la base de la programación extrema.

- Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento.
- Es necesaria la refactorización del código, ésta es la manera de mantener el código simple a medida que crece.
- Se simplifica la documentación: El código debe comentarse en su justa medida, intentando que el código esté autodocumentado. Para ello se deben elegir adecuadamente los nombres de las variables, métodos y clases.

Aplicando la simplicidad junto con la autoría colectiva del código y la programación por parejas se asegura que mientras más grande se haga el proyecto, todo el equipo conocerá más y mejor el sistema completo.

Comunicación: La comunicación se realiza de diferentes formas.

- Para los programadores el código comunica mejor mientras más simple sea. Si el código es complejo hay que esforzarse para hacerlo inteligible. El código autodocumentado es más fiable que los comentarios ya que éstos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse sólo aquello que no va a variar, por ejemplo, el objetivo de una clase o la funcionalidad de un método.
- Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad.
- Los programadores se comunican constantemente gracias a la programación por parejas.
- La comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo. El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

Retroalimentación (feedback):

- Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real.
- Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en los que es más importante.
- El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el estado de salud del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

Coraje o valentía: Hay que ser valiente para confiar en que la programación por parejas beneficia la calidad del código sin repercutir negativamente en la productividad. Se requiere coraje para enfrentar los cambios que el proyecto requiere.

3.2.2.2.2 Scrum

Scrum es un marco de trabajo iterativo e incremental que se emplea en entornos en los cuales los requisitos son inestables y requieren rapidez y flexibilidad.

Scrum está basado en los siguientes principios ágiles:

- Colaboración estrecha entre el equipo.
- Predisposición y respuesta al cambio.
- Prefiere el conocimiento de las personas al explícito de los procesos.
- Desarrollo incremental con entregables frecuentes.
- Comunicación verbal directa entre los implicados en el proyecto.

- Motivación y responsabilidad de los equipos por la auto-gestión, auto-organización y compromiso.

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos, conocidos como iteraciones. Se denomina “sprint” a cada iteración de desarrollo y se recomienda realizarlas con duraciones máximas de 30 días. La idea central es que un sprint se fije objetivos (funcionalidades que desarrollarán) al comienzo del mismo, y que el trabajo acordado esté finalizado al terminar el sprint.

ELEMENTOS DE LA METODOLOGÍA SCRUM:

Roles en el equipo de Scrum:

Product Owner: Representa a todos los interesados en el producto final. Hace las veces de cliente en el lugar.

Sus áreas de responsabilidad son:

- Definición/priorización de requisitos.
- Lanzamiento del proyecto.

Scrum Master: Es el responsable del seguimiento de las prácticas de Scrum.

Sus áreas de responsabilidad son:

- Debe velar porque se den las condiciones para trabajar, removiendo obstáculos.
- Formación y entrenamiento de la metodología.
- Garantía de cumplimiento de roles y responsabilidades.

Team Members: Responsables de generar un incremento del objetivo del proyecto. Diseñan, codifican y prueban las funcionalidades definidas para el sprint en curso.

El grupo debe tener las siguientes características:

- Auto-gestión
- Auto-organización
- Multi-funcional

La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro del equipo es mediante la conversación cara a cara.

Procedimiento Scrum:

El Product Owner es quien, sobre la base de los requerimientos planteados por clientes y usuarios, elabora la lista de requisitos, denominada product backlog, y les asigna prioridades.

Antes de comenzar un sprint, el Product Owner discute y define con el resto del equipo qué requisitos o ítems del product backlog habrá que desarrollar en el sprint, construyendo con ellos el sprint backlog. Se parte de requisitos para generar tareas. El sprint backlog, por lo tanto, no es un subconjunto del product backlog, ya que sus ítems son tareas, mientras que los de éste son requisitos. Ver **figura 10**.

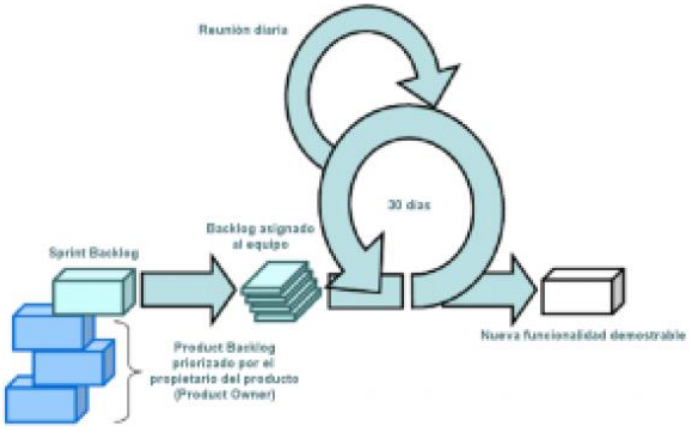


Figura 10. Procedimiento Scrum

En cuanto a la organización de los sprints, también existen algunas reglas.

- Durante un sprint, no se pueden cambiar los integrantes de un grupo de trabajo ni el sprint backlog. Lo único que se puede hacer es cancelar un sprint por razones de fuerza mayor.
- Durante el sprint, se realizan diariamente las scrum daily meetings, en las que participa todo el equipo, y en las que se analiza el avance y el trabajo del día. Estas reuniones son las que dieron nombre al método. El Scrum Master hace de moderador de estas reuniones.
- Al finalizar el sprint, se realiza una reunión denominada sprint review meeting, de la que se obtienen las lecciones aprendidas, que se dejan registradas en un artefacto denominado sprint retrospective. Ver **figura 11**.

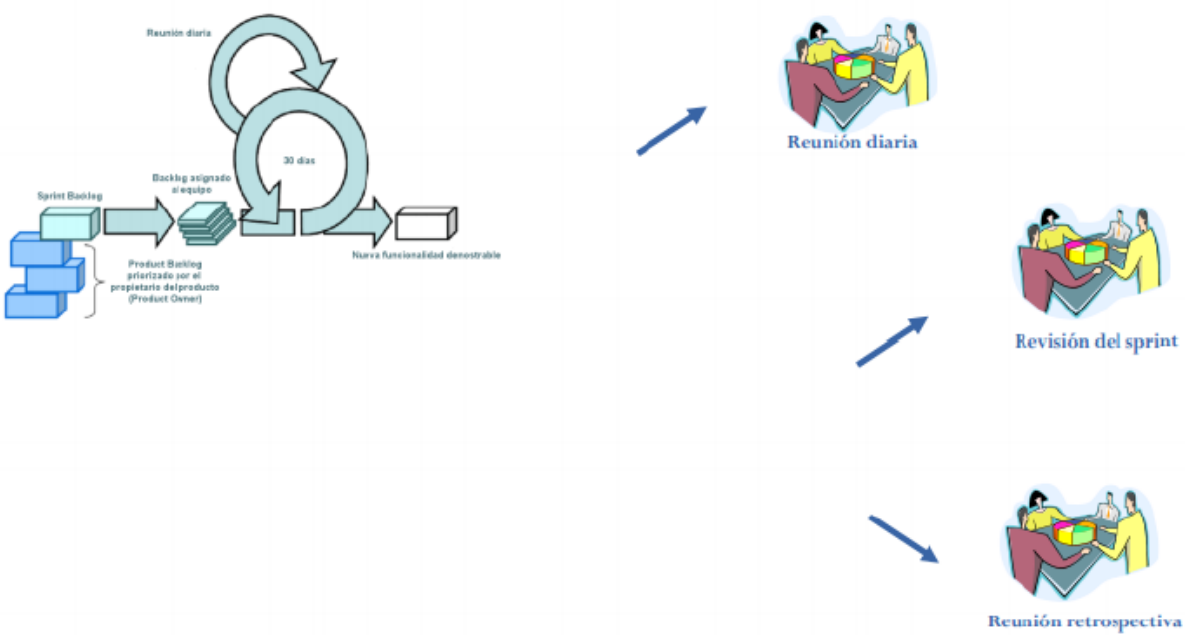


Figura 11. Reuniones y revisión del sprint en Scrum

En cuanto a las métricas, se pueden usar las que desee el equipo. No obstante, hay un reporte típico de Scrum, denominado burndown chart. El objetivo de este gráfico es hacer un seguimiento sobre la base del trabajo que falta por hacer. Se puede realizar durante el sprint, en cuyo caso se llama sprint burndown chart, y muestra día a día cuánto trabajo falta realizar dentro del sprint, y su relación con el que se esperaba realizar. También se puede proceder sprint a sprint el product burndown chart, en el nivel del proyecto.

Visto en forma más global, un proyecto Scrum tiene tres fases:

- Inicio: Incluye la planificación de una versión, con una primera estimación en tiempo y costo, y un diseño de alto nivel.
- Fase iterativa: Varios sprints.
- Cierre: Preparación para la versión desplegable, documentación final y entornos necesarios.

3.2.3 Estándares

Los estándares son un conjunto de criterios aprobados, documentados y disponibles para determinar la adecuación de una acción (estándar de proceso) o de un objeto (estándar de producto).

Los estándares son útiles porque:

- Agrupan lo mejor y más apropiado de las buenas prácticas y usos del desarrollo de software.
- Engloban los “conocimientos” que son patrimonio de una organización.
- Proporcionan un marco para implementar procedimientos de aseguramiento de la calidad.
- Proporcionan continuidad entre el trabajo de distintas personas.

- Sirven como referencia para definir procesos en una organización y para autoevaluación.

3.2.3.1 Capability Maturity Model Integration (CMMI)

CMMI surge de la evolución de CMM (ver CMM en **apéndice**). A finales de los 90 algunas organizaciones llevaban a cabo planes de calidad que integraban de forma simultánea varios modelos CMM. Para facilitar la incorporación de varios CMM's, el Software Engineering Institute (SEI) desarrolla y publica en 2001 el modelo CMMI que integra: CMM-SW, SE-CMM y IPD-CMM.

CMMI proporciona una guía para desarrollar procesos, que además ayuda a evaluar la madurez de la organización o capacidad de un área de procesos.

Madurez

Atributo de las organizaciones que desarrollan o mantienen los sistemas de software. En la medida que éstas llevan a cabo su trabajo siguiendo procesos, y en la que éstos se encuentran homogéneamente implantados, definidos con mayor o menor rigor; conocidos y ejecutados por todos los equipos de la empresa; y medidos y mejorados de forma constante, las organizaciones serán más o menos "maduras".

Capacidad

Atributo de los procesos. El nivel de capacidad de un proceso indica si sólo se ejecuta, o si también el proceso es planificado, se encuentra formalmente definido, se mide y se mejora de forma sistemática.

CMMI incluye los procesos de Ingeniería de Software e Ingeniería de Sistemas. Contiene 25 áreas de proceso. Cada área de proceso está formada por: objetivos específicos, prácticas específicas, objetivos genéricos, y prácticas genéricas.

El modelo CMMI está representado de forma continua y escalonada. La selección entre el modelo escalonado y el continuo depende del objetivo de la organización.

CMMI modelo continuo

Esta representación se centra en la mejora de un proceso o un conjunto de ellos relacionados a un área de proceso en que una organización desea mejorar, una organización puede obtener la certificación para un área de proceso en cierto **nivel de capacidad**.

Vistas desde la representación continua del modelo, las áreas de proceso se agrupan en 4 categorías según su finalidad: gestión de proyectos, Ingeniería, gestión de procesos y soporte a las otras categorías. Ver **tabla 1**.

Área de proceso	Categoría
Análisis y resolución de problemas	Soporte
Gestión de la configuración	Soporte
Análisis y resolución de decisiones	Soporte
Gestión integral de proyecto	G. Proyectos
Gestión integral de proveedores	G. Proyectos
Gestión de equipos	G. Proyectos
Medición y análisis	Soporte
Entorno organizativo para integración	Soporte
Innovación y desarrollo	G. Procesos
Definición de procesos	G. Procesos
Procesos orientados a la organización	G. Procesos
Rendimiento de los procesos de la organización	G. Procesos
Formación	G. Procesos
Integración de producto	Ingeniería
Monitorización y control de proyecto	G. Proyectos
Planificación de proyecto	G. Proyectos
Gestión calidad procesos y productos	Soporte
Gestión cuantitativa de proyectos	G. Proyectos
Desarrollo de requisitos	Ingeniería
Gestión de requisitos	Ingeniería
Gestión de riesgos	G. Proyectos
Gestión y acuerdo con proveedores	G. Proyectos
Solución técnica	Ingeniería
Validación	Ingeniería
Verificación	Ingeniería

Tabla 1. Áreas de proceso y categorías de CMMI en el modelo continuo

El modelo continuo de CMMI está formado por 6 niveles de capacidad del proceso:

0. Incompleto
1. Desempeñado
2. Administrado
3. Definido
4. Administrado cuantitativamente
5. Optimizado

Nota: Cada nivel de capacidad es construido sobre el nivel anterior. Un proceso alcanza un nivel de capacidad siempre y cuando haya alcanzado el nivel anterior.

CMMI modelo escalonado

El modelo escalonado, al igual que CMM, describe un camino evolutivo en 5 **niveles de madurez** de procesos, además integra nuevas áreas de proceso. Ver **tabla 2**.

Nivel de Madurez	Áreas de proceso
1- Inicial Madurez de un proceso caracterizada por resultados impredecibles.	Ninguna
2- Administrado Madurez del proceso caracterizada por el desempeño repetible del proyecto. El foco clave del proceso está en actividades y prácticas a nivel proyecto.	• Administración de Requerimientos. • Planeación de proyectos. • Monitoreo y control de proyectos. • Administración del acuerdo con el proveedor. • Administración de configuración. • Aseguramiento de calidad de proceso y producción. • Mediciones y análisis.
3- Definido Madurez del proceso caracterizada por mejorar el desempeño del proceso dentro de una organización.	• Desarrollo de requerimientos. • Solución técnica. • Integración de producto. • Verificación, validación. • Enfoque organizacional en proceso. • Definición de procesos de la organización. • Capacitación organizacional. • Administración integral de proyectos. • Administración de riesgo. • Análisis de decisión y resolución.
4- Administrado cuantitativamente Caracterizada por mejorar el desempeño organizacional.	• Desempeño de procesos organizacionales. • Administración cuantitativa de proyectos.
5- Optimizado Madurez del proceso caracterizada por un desempeño organizacional rápido y configurable, mejora continua y cuantitativa de procesos.	• Innovación y despliegue organizacional. • Análisis causal y resolución.

Tabla 2. CMMI modelo escalonado

El modelo CMM y el modelo CMMI se diferencian básicamente en que el primero se enfoca principalmente a las organizaciones o áreas de tecnologías de información en cambio el modelo CMMI como su nombre lo indica es un modelo integrado y mejorado que se puede aplicar a un número mayor de organizaciones de diferentes sectores.

3.2.3.2 MoProSoft

El modelo de procesos para la industria de software mexicana (MoProSoft), fue desarrollado pensando en facilitar a las organizaciones dedicadas al desarrollo y mantenimiento de software la adopción de las mejores prácticas reconocidas internacionalmente a través de modelos como SW-CMM, CMMI, TSP, PSP, PMBOK y SWEBOK.

Estructura de MoProSoft

MoProSoft es un modelo integrado que sintetiza las mejores prácticas en un conjunto pequeño de procesos que abarcan las responsabilidades asociadas a la estructura de una organización que son: la alta dirección, gestión y operación. Ver **figura 12**.

Las salidas de un proceso están claramente dirigidas como entradas a otros; las prácticas de planeación, seguimiento y evaluación se incluyeron en todos los procesos de gestión y administración; por su parte los objetivos, los indicadores, las mediciones y las metas cuantitativas fueron incorporados de manera congruente y práctica en todos los procesos; las verificaciones, validaciones y pruebas están incluidas de manera explícita dentro de las actividades de los procesos; y existe una base de conocimientos que resguarda todos los documentos y productos generados. Ver **tabla 3**.

A continuación se muestra la estructura y el propósito de cada uno de los procesos de MoProSoft:

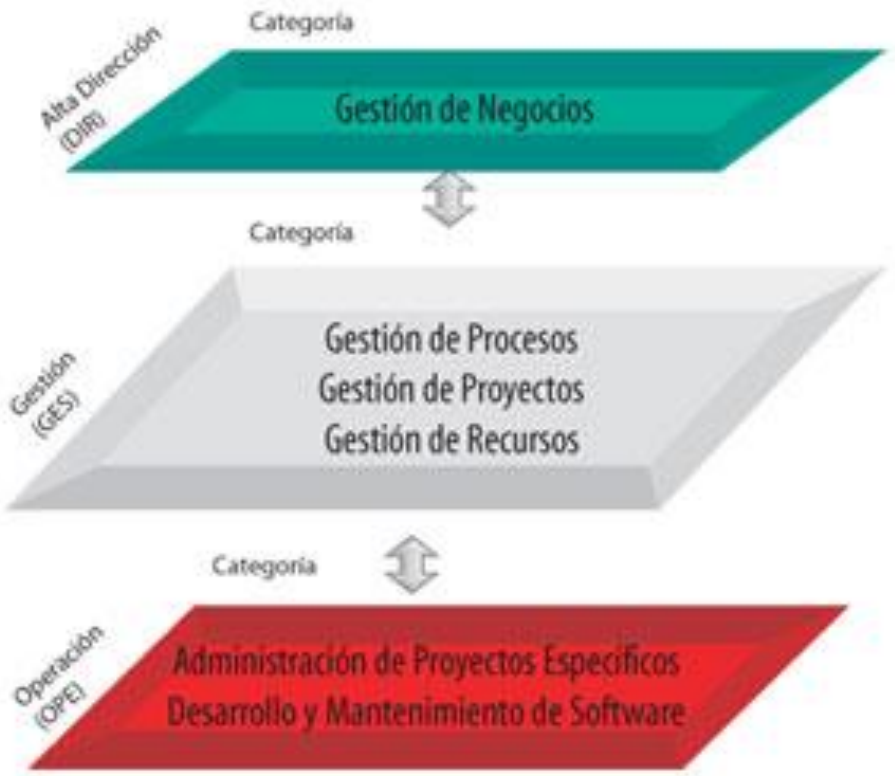


Figura 12. Estructura de MoProSoft

Categoría	Proceso	Propósito
Alta dirección	Gestión de negocio	Establecer la razón de ser de la organización, sus objetivos y las condiciones para lograrlos, para lo cual es necesario considerar las necesidades de los clientes, así como evaluar los resultados para poder proponer cambios que permitan la mejora continua. Adicionalmente habilita a la organización para responder a un ambiente de cambio y a sus miembros para trabajar en función de los objetivos establecidos.
Gestión	Gestión de procesos	Establecer los procesos de la organización, en función de los procesos requeridos identificados en el plan estratégico. Así como definir, planificar e implantar las actividades de mejora en los mismos.
Gestión	Gestión de proyectos	Asegurar que los proyectos contribuyan al cumplimiento de los objetivos y estrategias de la organización.
Gestión	Gestión de recursos	Conseguir y dotar a la organización de los recursos humanos, infraestructura, ambiente de trabajo y proveedores, así como crear y mantener la base de conocimiento de la organización. La finalidad es apoyar el cumplimiento de los objetivos del plan estratégico de la organización. Las actividades de este proceso se apoyan en tres subprocesos: - Recursos humanos y ambiente de trabajo. - Bienes, servicios e infraestructura. - Conocimiento de la organización.
Operación	Administración de proyectos específicos	Establecer y llevar a cabo sistemáticamente las actividades que permitan cumplir con los objetivos de un proyecto en tiempo y costo esperados.
Operación	Desarrollo y mantenimiento de software	Realización sistemática de las actividades de análisis, diseño, construcción, integración y pruebas de productos de software nuevos o modificados cumpliendo con los requerimientos especificados.

Tabla 3. Estructura y propósito de MoProSoft

3.2.3.3 ITIL

ITIL (Information Technology Infrastructure Library) es un estándar que busca mejorar la calidad de servicio y el desarrollo eficaz y eficiente de los procesos que cubren las actividades más importantes de las organizaciones en sus sistemas de información y tecnologías de información.

ITIL proporciona un conjunto de buenas prácticas de dirección y gestión de servicios de tecnologías de la información en lo referente a personas, procesos y tecnología, estas buenas prácticas fueron extraídas por la Oficina Gubernativa de Comercio Británica (OGC, Office of Goverment Comerce) de organismos punteros del sector público y privado a nivel internacional. Este framework o marco de procesos es utilizado por cientos de organizaciones en el mundo y ha sido desarrollado reconociendo la dependencia creciente que tienen éstas en la tecnología para alcanzar sus objetivos.

La realización de las buenas prácticas especificadas en ITIL hace posible que los departamentos y organizaciones puedan reducir costes, mejorar la calidad del servicio tanto de clientes externos como internos y aprovechar al máximo las habilidades y experiencia del personal, mejorando su productividad.

La **figura 13** mostrada a continuación, explica a grandes rasgos los diversos componentes que cubre ITIL.

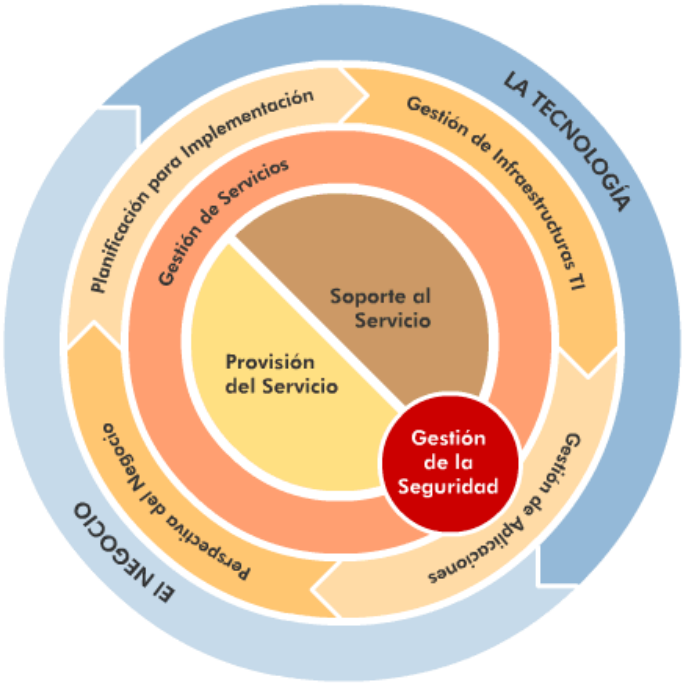


Figura 13. Componentes de ITIL

De acuerdo a este diagrama es de resaltar que ITIL cubre aspectos generales que van desde el conocimiento del negocio hasta la planeación e implementación de los recursos tecnológicos, a través de una gestión que focaliza la infraestructura de tecnologías de Información (TI) y el desarrollo de aplicaciones, ocupando siempre un lugar preferencial la seguridad de la información.

Gestión de servicios

El objetivo principal de la gestión de servicios es asegurar que los servicios IT (Information Technology) están alineados con las necesidades actuales y futuras del negocio y sus clientes.

Los procesos de gestión de servicios son el corazón de ITIL y pueden subdividirse en dos áreas. Ver **figura 14**.

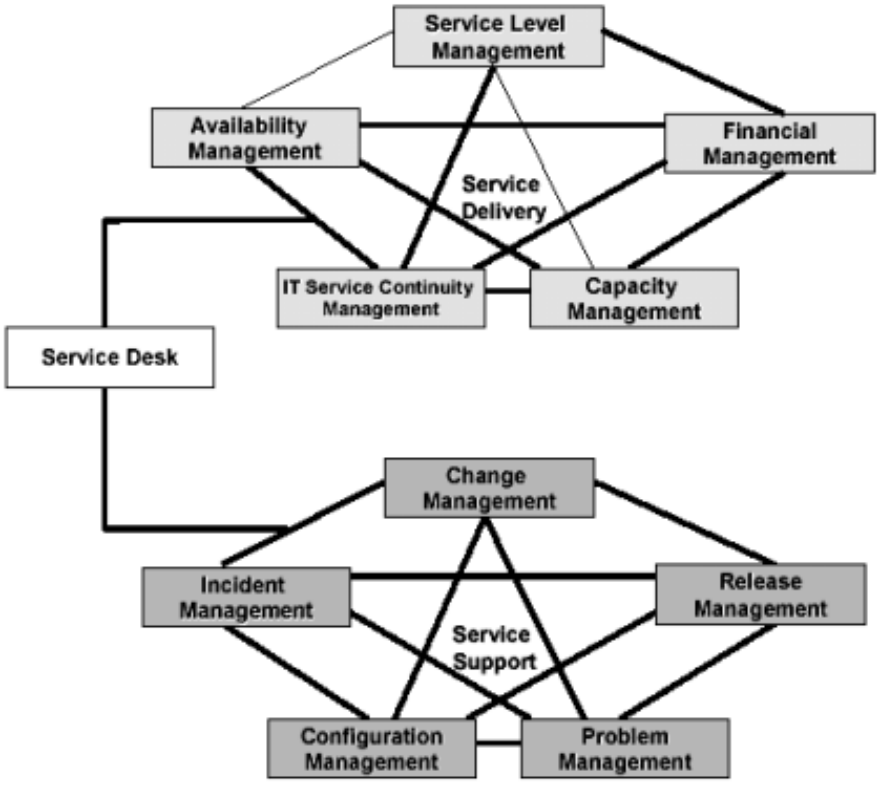


Figura 14. Áreas de la gestión de servicios de ITIL

Deliver IT services (provisión de servicios IT)

La entrega de servicios cubre todos los procesos tácticos necesarios para una entrega con calidad y a un costo efectivo de los servicios de TI. La calidad de un servicio debe ser cuantificable y alcanzable, esto se logra a través del establecimiento de acuerdos del nivel de cada servicio y están sustentados en la información que proporcionan otros procesos tales como la capacidad de los recursos que están involucrados en los servicios, la disponibilidad calculada y la medida, con sus relaciones entre sí y las causas de incumplimiento en tal caso que se presente.

Abarca los siguientes procesos:

- Service level management (gestión de los niveles de servicio).
- Capacity management (gestión de la capacidad).
- Financial management (gestión financiera de los servicios TI).
- IT serivice continuity management (gestión de la continuidad).
- Availability management (gestión de la disponibilidad).
- Service desk (gestión de las relaciones con el cliente).

Support IT services (soporte de servicios)

Los proveedores de servicios deben ofrecer a los usuarios un adecuado soporte. El soporte de servicios definido por ITIL ayuda a la empresa a gestionar el hardware, software y los recursos humanos para asegurar la continuidad y no interrupción de los servicios ofrecidos por el negocio.

El soporte de servicios se ocupa de la planificación a largo plazo y del perfeccionamiento de la provisión de estos servicios. En este segmento de procesos, la preocupación está dirigida al acceso que tienen los usuarios a los servicios que soportan sus funciones de negocio. Los usuarios de cada uno de los servicios de TIC que se entregan en la organización requieren de un soporte para su correcta operación y atención cuando se producen fallos. Este elemento abarca:

- Service desk (gestión de las relaciones con el cliente).
- Incident management (administración de incidentes).
- Problem management (administración de problemas).
- Release management (administración de versiones de software).
- Change management (administración de cambios).
- Configuration management (administración de la configuración).

Administrar las aplicaciones

Ofrece un conjunto de buenas prácticas para la gestión de todo el ciclo de vida de las aplicaciones, centrándose sobre todo en la definición de requisitos e implementación de soluciones.

Gestión de la seguridad

La gestión de seguridad definida por ITIL, explica los procesos de gestión de la seguridad con la gestión de servicios TI. Esta guía se centra en el proceso de implementación de los requisitos de seguridad identificados en los acuerdos de niveles de servicio TI más que en la consideración de las políticas de seguridad para el negocio.

Comprende los siguientes 10 elementos del código de prácticas para la administración de la seguridad de la información:

1. Políticas de seguridad: Proporciona a la alta dirección apoyo para la seguridad de la información.
2. Organización de la seguridad: Ayuda a administrar la seguridad de la información dentro de la organización.
3. Clasificación y control de activos: Provee las medidas de seguridad necesarias para proporcionar una protección adecuada a los activos de la organización.

4. Seguridad del personal: Necesaria para reducir los riesgos de errores humanos, robo, fraude o mal uso de las instalaciones y equipo.
5. Seguridad física y ambiental: Previene el acceso físico no autorizado a los sistemas de información, así como posibles daños a las instalaciones y a la información del negocio.
6. Administración de comunicaciones y operaciones: Permite garantizar la operación correcta y segura de las instalaciones de procesamiento de la información.
7. Control de acceso: Previene el acceso lógico y cambio, no autorizado, a la información y a los sistemas de información, otorgando confidencialidad en la información, para evitar interrupciones en los procesos normales de producción.
8. Desarrollo y mantenimiento de los sistemas: Permite incorporar la seguridad a los sistemas de información.
9. Administración de la continuidad del negocio: Contrarresta las interrupciones a las actividades del negocio y protege los procesos críticos del negocio contra los efectos causados por fallas mayores o desastres.
10. Conformidad: Contribuye a evitar infracciones a las leyes civiles, jurídicas, obligaciones reguladoras o contractuales y cualquier otro requerimiento de seguridad.

Planeación para la Implementación de la Administración de Servicio

Explica los pasos necesarios para identificar como una organización puede esperar beneficiarse de ITIL, y que hacer para obtener estos beneficios. Esta publicación cubre los temas y actividades involucradas en planeación, implementación y mejora de los procesos de administración de servicios dentro de una organización.

Procesos tratados en esta publicación:

- Proceso de mejora continua.

Perspectiva del Negocio

Estos procesos están orientados a ayudar a los administradores del negocio a entender la provisión de servicios de TIC (Technology Information Center). Abarca lo siguiente:

- Administración de la continuidad del negocio.
- Alianzas y outsourcing.
- Sobrevivir al cambio.
- Alineación del negocio y TIC.

Gestión de la infraestructura

La gestión de la infraestructura cubre:

- Gestión del servicio de red.
- Gestión de las operaciones.
- Gestión de los procesadores locales.
- Aceptación e instalación de las computadoras.
- Gestión de los sistemas.

3.2.3.4 COBIT

COBIT (Control Objectives for Information and related Technology) es un estándar publicado en 1996 por el ITGI (Instituto de Control de TI) y la ISACA (Asociación de Auditoría y Control de Sistemas de Información).

El objetivo principal del proyecto COBIT es el desarrollo de políticas claras y buenas prácticas para la seguridad y el control de tecnología de información.

La última versión de COBIT (COBIT5) clasifica 36 procesos de negocio relacionados con las tecnologías de la información en 2 áreas de conocimiento y 5 dominios:

Área de conocimiento: Gobierno corporativo de TI

- Evaluar, dirigir y monitorear (EDM) – 5 procesos

Área de conocimiento: Administración de TI corporativa

- Alinear, planear, organizar (PO) – 12 procesos
- Construcción, adquisición e implementación (BAI) – 8 procesos
- Entrega, servicio y soporte (DSS) – 8 procesos
- Monitoreo, evaluación e informes (MEI) – 3 procesos

Para cada uno de los procesos definidos en los cinco dominios de COBIT5, se define un objetivo de control. Podemos definir “control” como las políticas, procedimientos, prácticas y estructuras organizacionales que han sido diseñadas para asegurar razonablemente que los objetivos del negocio se alcanzarán y los eventos no deseados, serán prevenidos o detectados y corregidos.

Estos objetivos de control de TI proporcionan un conjunto completo de requerimientos de alto nivel a considerar por la gerencia para un control efectivo de cada proceso de TI. Estos controles son sentencias de acciones de gerencia que deben de aumentar el valor o reducir el riesgo en el negocio, generalmente consisten en políticas, procedimientos, prácticas y estructuras organizacionales, las cuales proporcionan un aseguramiento razonable de que los objetivos del negocio se verán alcanzados.

COBIT5 proporciona la siguiente clasificación para valorar la madurez de los procesos de la organización con objeto de conseguir una mejora continua:

- Nivel 0: Incompleto.
- Nivel 1: Realizado.
- Nivel 2: Gestionado.
- Nivel 3: Establecido.
- Nivel 4: Predecible.
- Nivel 5: Optimizado.

3.2.4 Herramientas

Las herramientas dan soporte a las capas anteriores centrándose en la automatización de algunas de las actividades manuales.

Las herramientas se pueden utilizar para automatizar las siguientes actividades:

- Actividades de gestión de proyectos.
- Métodos técnicos usados en la ingeniería del software.
- Soporte de sistemas general.
- Marcos de trabajo para otras herramientas.

La automatización ayuda a eliminar el tedio del trabajo, reduce las posibilidades de errores, y hace más fácil usar buenas prácticas de Ingeniería de Software.

3.2.4.1 BPMN

BPMN (Business Process Model and Notation) es una notación gráfica que describe la lógica de los pasos de un proceso de negocio. Esta notación ha sido especialmente diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades.

BPMN es un estándar internacional de modelado de procesos aceptado por la comunidad (OMG) independiente de cualquier metodología de modelado de procesos que proporciona un lenguaje común para que las partes involucradas puedan comunicar los procesos de forma clara, completa y eficiente. De esta forma BPMN define la notación y semántica de un diagrama de procesos de negocio (Business Process Diagram, BPD).

- BPD es un diagrama diseñado para representar gráficamente la secuencia de todas las actividades que ocurren durante un proceso, basado en la técnica de “flow chart”, incluye además toda la información que se considera necesaria para el análisis.
- BPD es un diagrama diseñado para ser usado por los analistas, quienes diseñan, controlan y gestionan procesos.
- Dentro de un diagrama de procesos de negocio BPD se utiliza un conjunto de elementos gráficos, agrupados en categorías, que permite el fácil desarrollo de diagramas simples y de fácil comprensión.

3.2.4.2 UML

UML no es una metodología o proceso ni está vinculada específicamente a alguna herramienta. UML es un lenguaje que nos permite construir modelos.

Modelo: Es una descripción analógica para ayudar a visualizar algo que no se puede observar directamente, se realiza con un propósito determinado y se destina a un público específico.

El software necesita modelos por las mismas razones que cualquier otra construcción humana: Para comunicar de manera sencilla una idea abstracta, existente o no, o para describir un producto existente.

Según varias definiciones, el software es en sí un modelo de la realidad. Si así fuera, un modelo de software es el modelo de un modelo; es decir un meta-modelo. El modelo más detallado de un producto de software es el código fuente. Pero es como

decir que el mejor modelo de un edificio es el edificio mismo; esto no nos sirve para concebirlo antes de la construcción ni para entender sus aspectos más ocultos con vistas al mantenimiento.

UML utiliza diagramas para mostrar distintos aspectos de un sistema. Un mismo sistema puede tener varios modelos, que dependen del propósito y del público al que se dirige.

UML puede utilizarse de dos formas:

- Como herramienta de comunicaciones entre humanos.
- Como herramienta de desarrollo.

UML trabaja con 13 tipos de diagramas para definir modelos estáticos o estructurales y modelos dinámicos o de comportamiento:

Los diagramas estáticos o estructurales de UML son:

- Diagrama de casos de uso.
- Diagrama de objetos (estático).
- Diagrama de clases.
- Diagrama de paquetes.
- Diagrama de componentes.
- Diagrama de despliegue.
- Diagrama de estructuras compuestas.

Los diagramas de dinámicos o de comportamiento son:

- Diagrama de secuencia.
- Diagrama de comunicación (o de colaboración).
- Diagrama de máquina de estados.
- Diagrama de actividades.
- Diagrama de visión global de la interacción.
- Diagrama de tiempos.

3.2.4.3 GQM

GQM (Goal Question Metric) es un paradigma para desarrollar y mantener un programa de métricas que ayudan a:

- Alinear las métricas con los objetivos del negocio de la organización y las metas técnicas.
- Mejorar el proceso del desarrollo de software.
- Mejorar la calidad del producto obtenido.

El principio básico que subyace tras la metodología GQM es que la medición debe ser realizada orientada a un objetivo.

GQM es una estructura jerárquica que define un objetivo, refina este objetivo en preguntas para entender sus componentes y finalmente define métricas que intentan dar información para responder a estas preguntas. Las preguntas ayudan a medir si se está alcanzando el objetivo definido, y por lo tanto solo se considerarán preguntas que sean potencialmente medibles. Ver **figura 15**.

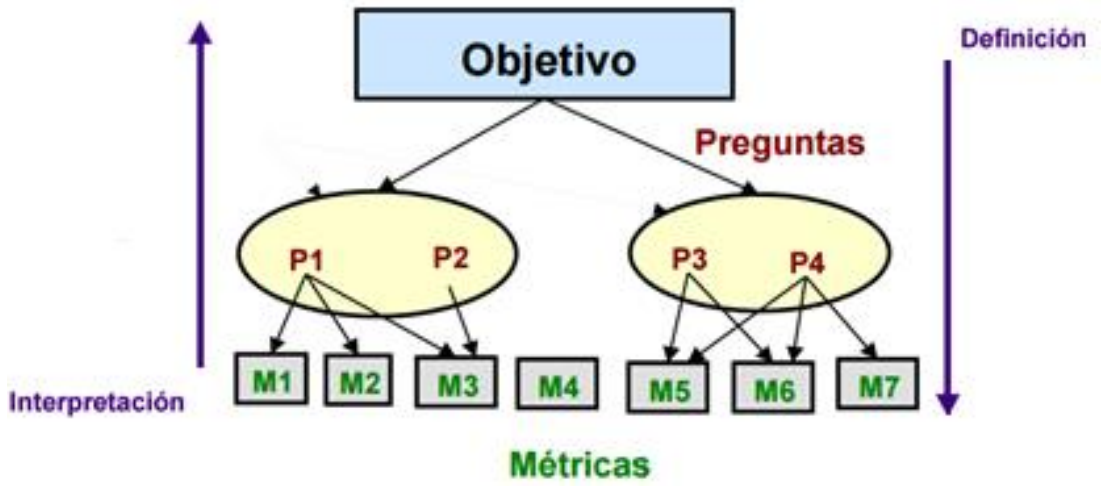


Figura 15. Principio básico de la metodología GQM

La metodología GQM sigue un proceso en cuatro fases; planificación, definición, recopilación de datos e Interpretación. Ver **figura 16**.

- En la fase de planificación se selecciona, define, caracteriza y planifica el proyecto para la aplicación de la medición, obteniéndose el plan de proyecto.
- En la fase de definición se define y documenta el programa de medición (objetivos, preguntas, métricas e hipótesis).
- La fase de recopilación de datos es en la que se reúnen los datos reales para ejecutar la medición.
- La fase de interpretación es en la que se procesan los datos recopilados respecto a las métricas definidas en forma de resultados de medición, que proporcionan respuestas a las preguntas planteadas en la fase de definición y a partir de aquí evaluar el logro del objetivo planteado.

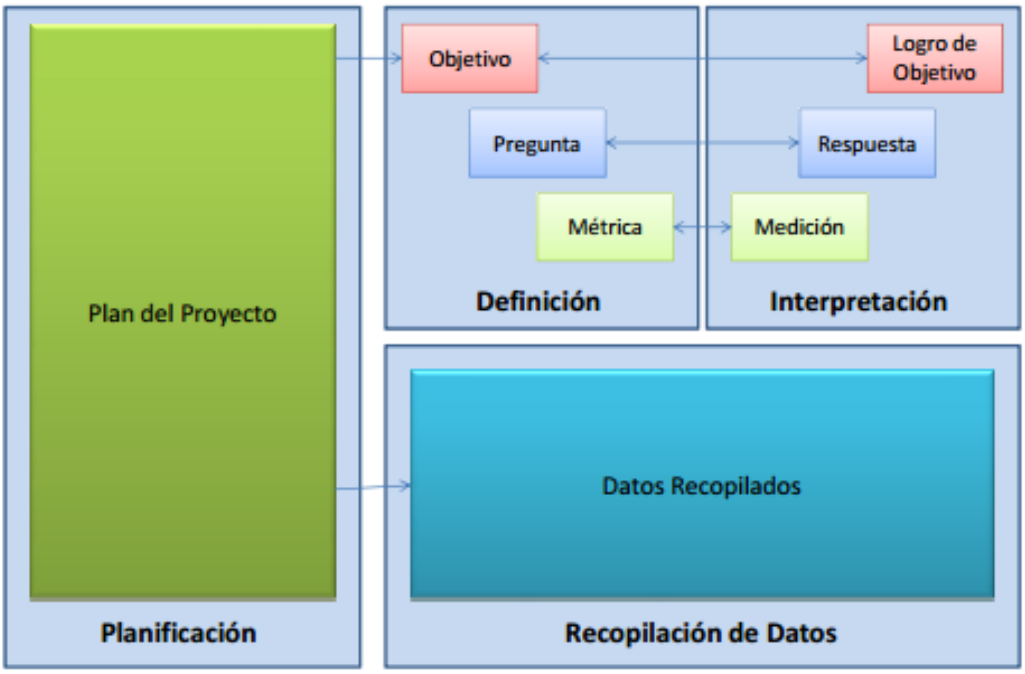


Figura 16. Proceso de la metodología GQM

3.3 Referencias:

Referencias bibliográficas:

[5] Ingeniería del Software, Un enfoque práctico. Roger S. Pressman. McGraw-Hill. Séptima Edición. 2010

[6] Ingeniería del Software. Un enfoque desde la guía Swebok. Sánchez, Salvador, Sicilia, Miguel Ángel y Rodrígez, Daniel. Alfaomega Grupo Editor. Primera Edición. México 2012.

[7] Ingeniería del Software. Sommerville, Ian. Pearson Educación. Séptima Edición. Madrid. 2006.

[8] IEEE Std. 610.12-1990 Glosary of Software Engineering Terminology

[9] UML Modelado de Software para profesionales. Fontela Carlos. Alfaomega Grupo Editor. Primera Edición. Buenos Aires. 2011.

[10] Guía del PMBOK. Project Management Institute. Cuarta edición. 2008.

[11] Administración de proyectos de informática. Francisco J. Toro López. Ecoe Ediciones. Primera Edición. Bogotá. 2013.

Referencias web:

[12] IBM Rational Software
http://www.ibm.com/software/rational

[13] Sitio web Software Guru Procesos de Software
http://sg.com.mx/content/view/23

[14] Roles en el desarrollo de Software
http://www.ganimides.ucm.cl/ygomez/descargas/Sist_inf2/apuntes/2009/Roles_desarrollo_software.pdf

[15] Manifiesto por el Desarrollo Ágil del Software

http://agilemanifesto.org/iso/es/manifesto.html

[16] Manifiesto por el Desarrollo Ágil del Software

https://www.nyce.org.mx/?s=moprosoft

Capítulo 4

Identificar áreas de oportunidad para la mejora de procesos

4.1 Identificar áreas de oportunidad para el desarrollo de software en la CSC del IINGEN UNAM

Tomando como base el proceso básico del ciclo de vida del software (toma de requerimientos, análisis, diseño, construcción, pruebas y mantenimiento) y partiendo del supuesto de que cada una de las etapas tienen la misma importancia, entonces, podemos decir que un proyecto de software debe cubrir cada una de ellas y por tanto, la sumatoria en porcentaje del total de las mismas debe ser igual al 100% (cada práctica representa un 16.6% del total del proyecto en un escenario ideal y homogéneo). Este escenario ideal se encuentra representado con color rojo en la **figura 17**.

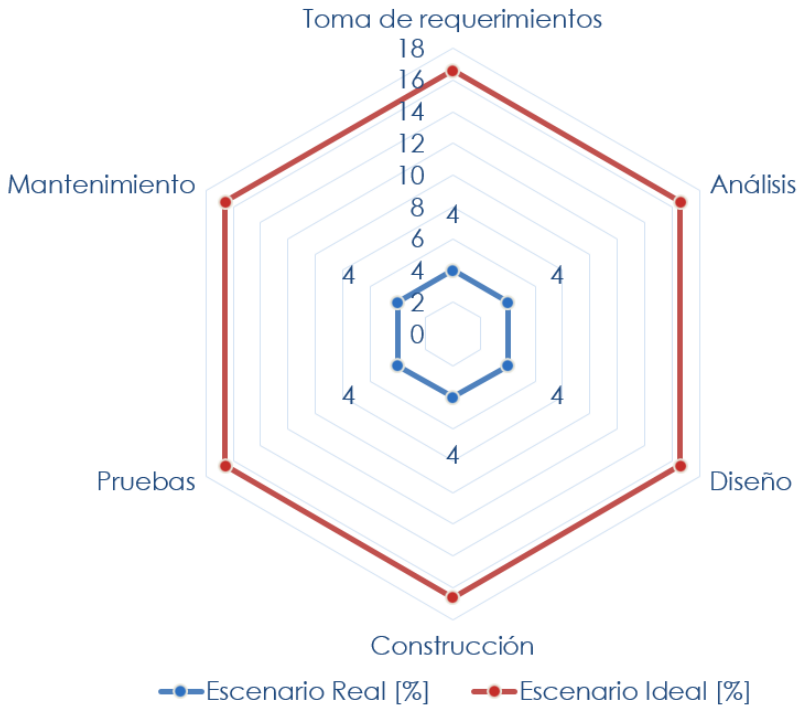


Figura 17. Porcentajes de las etapas del ciclo de vida del software

De acuerdo a la **figura 17**, también se puede apreciar una zona marcada en color azul, la cual representa la situación actual en la que se encuentra el equipo o grupo de trabajo de la CSC del IINGEN con respecto a cada una de las etapas del ciclo de vida del software. Para determinar y asignar el porcentaje en azul de la gráfica anterior, lo primero que se realizó fue identificar las deficiencias actuales, que es donde se encuentran las áreas de oportunidad o de mejora:

Áreas de oportunidad (problemas actuales)

1. Escasa documentación de los sistemas creados.
2. Diversas maneras de codificar utilizadas por cada uno de los desarrolladores.
3. No se cuenta con un repositorio de respaldo y de manejo de versiones.
4. No existe una práctica sana para llevar acabo la toma de requerimientos, tanto de nuevos sistemas como de los ya existentes que requieren de actualizaciones en cuanto a funcionalidad.

De acuerdo a la ley de Pareto (ver **apéndice**), el 80% de los beneficios de contar con una mejora de proceso en el desarrollo de software se concentran en las áreas de oportunidad de mejora identificadas anteriormente (en adelante conocidas en este trabajo de tesis como las 4 alfas). Si a cada área de oportunidad se le asignara un porcentaje en este caso del 20%, tendríamos en la sumatoria total 80% que nos haría aproximarnos a la zona de interés del gráfico que es la parte roja.

Las cuatro Alfas (α): $\alpha = 20\%$

1. Documentación
2. Normalizar código
3. Repositorio y manejo de versiones
4. Requerimientos.

$4\alpha = 80\%$

Con lo anterior se puede apreciar el problema a resolver de este trabajo de tesis: Lograr un estándar para la realización del software, garantizar la transferencia de conocimientos generados dentro de la Coordinación de Sistemas de Cómputo del Instituto de Ingeniería, tener un control eficiente de versiones y un repositorio en común. Esto se llevará a cabo trabajando sobre las cuatro α's planteadas.

4.2 Identificar la trazabilidad de los sistemas de información

Si partimos de la experiencia del equipo de Ingeniería de Software en el desarrollo de sistemas, se puede considerar que existen varios elementos en común, es decir, hay ciertas funcionalidades que se repiten en la mayoría de los sistemas desarrollados en el IINGEN. A estas funcionalidades en común les daremos el nombre de trazabilidad. Ver **figura 18**.

Figura 18. Gráfico de trazabilidad

- Monitoreo: Son los elementos que nos permiten detectar y llevar un registro o historial de las acciones que los usuarios realizan dentro de un sistema. Se almacenan datos como fechas, nombres, entre otros, de los responsables de dichas acciones.
- Interfaz de usuario (UI): Es la parte de un sistema que interactúa con el usuario.
- Comunicación: Son los elementos que nos permiten llevar a cabo la interacción entre dos o más sistemas o plataformas.
- Lógica de Negocio: Son los algoritmos que dan la funcionalidad al sistema. Este elemento contempla el almacenamiento de la información en bases de datos, en archivos o en algún otro repositorio.
- Seguridad: Este elemento abarca desde el control de acceso de los usuarios a un sistema, hasta la protección misma de la información, con la finalidad de poder garantizar la accesibilidad, disponibilidad e integridad de la misma.

La importancia de la trazabilidad radica en que es posible analizar, diseñar y construir un framework que contemple los aspectos antes mencionados de manera genérica y de acuerdo a cada nuevo sistema de software desarrollado en el IINGEN adaptarlo a sus requerimientos. Esto propiciaría que todos los desarrollos estén fundamentados en la misma trazabilidad, obteniendo con esto beneficios en tiempos, no solo de desarrollo sino de mantenimiento, adaptabilidad y costos. Adicionalmente, podemos garantizar la transferencia de conocimiento ya que lo que se busca es documentar dicha traza y así cualquier miembro del equipo de Ingeniería de Software tendrá la capacidad de desarrollar nuevos sistemas, dar mantenimiento y hacer adecuaciones a los desarrollos ya existentes.

La trazabilidad del sistema se encuentra dividida y acotada, por lo que es posible implementar nuevas funcionalidades, algoritmos, diseños, etc. sin afectar a los sistemas o funcionalidades de los mismos. También es posible lograr la interoperabilidad entre sistemas y plataformas, e inclusive es factible poder emplear lo mejor de cada una de las tecnologías existentes y las que surjan en un futuro.

La mejora de procesos (madurez) en el Instituto de Ingeniería se realizará en varias etapas. Con el presente trabajo de tesis se busca cumplir con la primera de ellas, sentando las bases necesarias para poder llevar a cabo las siguientes, como se verá en el capítulo siguiente.

Capítulo 5

Plan de mejora de procesos

5.1 Metodología propuesta

En el capítulo 4 se identificaron las cuatro α's (requerimientos, documentación, normalización de código, repositorio y manejo de versiones) sobre las cuales se trabajará para lograr una mejora de procesos en el desarrollo de software dentro de la Coordinación de Sistemas de Cómputo del Instituto de Ingeniería de la UNAM.

Para entrar en un proceso de mejora continua, se propone una metodología de cinco pasos (ver **figura 19**) que busca en primera instancia ayudar a poner orden en el desarrollo de software sin necesidad de implementar una o varias metodologías ya establecidas como podrían ser RUP para el caso tradicional o SCRUM para el caso ágil. Por el momento el implementar alguna de las metodologías ya establecidas podría entorpecer el desarrollo de los sistemas, pues es recomendable iniciar por etapas y la primera de ellas es partir de lo que se conoce y es de dominio común para la organización, así como obtener elementos de valor que puedan ser utilizados y probados en proyectos para así iniciar con el proceso de mejora en un ambiente real.

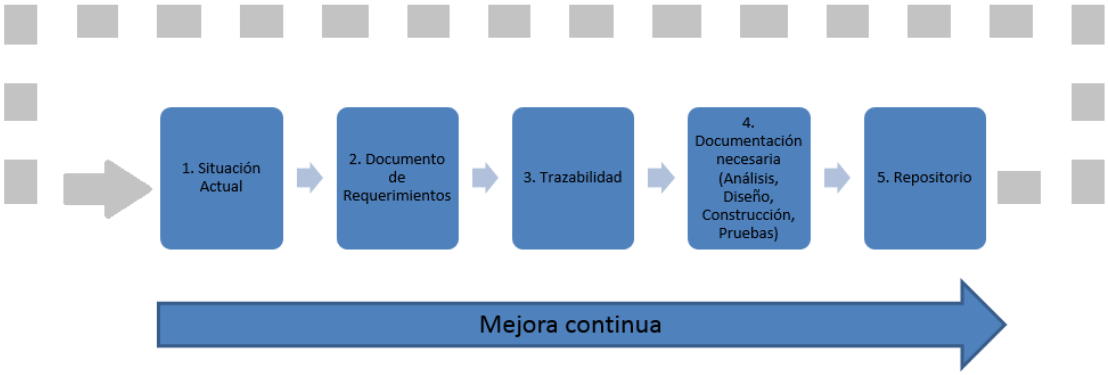


Figura 19. Metodología propuesta

5.2 Pasos a seguir para lograr la mejora de procesos para el desarrollo de software

De acuerdo a la metodología propuesta, son cinco pasos los que se deben considerar para lograr una mejora continua en el proceso de desarrollo de software y fortalecer las cuatro **α's**:

1. **Situación actual**: Considerando como referencia el principio básico del paradigma GQM, se formularán cuestionarios de evaluación (scripts de evaluación) con la finalidad de que sean resueltos por los integrantes del grupo de trabajo contando así con una visión más real de nuestro escenario actual. Los scripts de evaluación se elaborarán con base en la biblioteca de ITIL (mejora de los procesos, alinear los procesos, reducir riesgos, garantizar el servicio).

Scripts de evaluación

Toma de requerimientos

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Se cuenta con documentos estandarizados de requerimientos?	No	0
2	¿Se cuenta con control y seguimiento de acuerdos?	Si	2.37
3	¿Existe una guía definida para la toma de requerimientos?	No	0
4	¿Se modelan los procesos?	No	0
5	¿Se administran los requerimientos?	Si	2.37
6	¿Se cuenta con un repositorio de requerimientos?	No	0
7	¿Se realizan Casos de uso de negocio?	No	0

Total: 4.74 %

Análisis

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Existen documentos de Arquitectura?	No	0
2	¿Existen Casos de uso de sistema?	No	0
3	¿Existe documentación estandarizada?	No	0
4	¿Se cuenta con repositorio de documentos de análisis?	No	0

Total: 0 %

Diseño

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Existen Diagramas de clases?	No	0
2	¿Existen Diagramas de modelado de BD?	No	0
3	¿Existe documentación estandarizada?	No	0
4	¿Se cuenta con repositorio de documentos de Diseño?	No	0

Total: 0 %

Construcción

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Existe Normalización de código?	No	0
2	¿Existe administración de integración?	No	0
3	¿Se cuenta con repositorio de código fuente y control de versiones?	Si	5.53

Total: 5.53 %

Pruebas

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Existe especificación de casos de prueba?	No	0
2	¿Existe estandarización de pruebas?	No	0
3	¿Se cuenta con repositorio para casos de prueba?	No	0

Total: 0 %

Mantenimiento

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Se administran los cambios y actualizaciones?	No	0
2	¿Se administran las incidencias?	No	0
3	¿Se tiene control de versiones?	No	0
4	¿Se cuenta con notas técnicas?	Si	4.15

Total: 4.15 %

En esta primera etapa de implementación dichos scripts se resolvieron en consenso con los integrantes del equipo de desarrollo del IINGEN obteniendo con ello el gráfico de la **figura 20**.

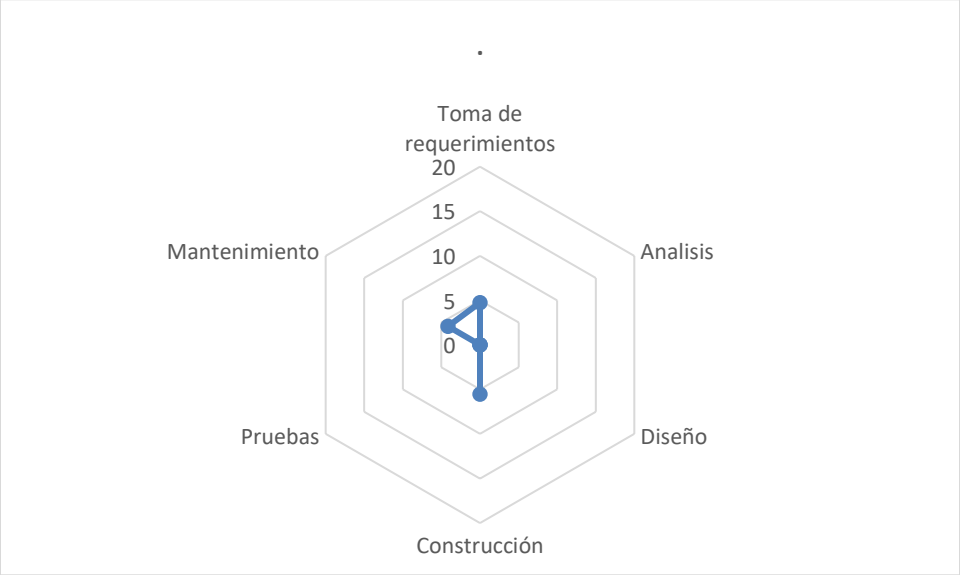


Figura 20. Practicas generales de Ingenierías de Software situación actual

2.- Documento de requerimientos: Como parte fundamental de la resolución de problemas, es necesario comprender el caso en cuestión, por lo que el determinar y estandarizar la documentación para la toma de requerimientos será fundamental para lograr un proyecto de software con mayor éxito. Con este paso se cubre una de las alfas (**α = Requerimientos**).

3.- Trazabilidad: Uno de los principales argumentos que se tienen en los grupos de desarrollo de software, es que nunca hay tiempo para documentar, por lo que la metodología propone encontrar la trazabilidad de los sistemas que se desarrollan en la organización y así identificar los elementos que convergen en la mayoría de ellos para que puedan ser analizados, diseñados y documentados. De esta manera el esfuerzo se haría una vez y se emplearía en múltiples ocasiones, al menos la base de los sistemas estaría uniforme y así se podría tener un mayor control y transferencia de conocimiento de los mismos. Con este paso se cubre una de las alfas (**α = Normalización de código**).

4.- Documentación necesaria: Como parte complementaria del paso tres, el tener la documentación de la arquitectura, análisis, diseño, pruebas, mantenimiento, control de versiones, etc. es de gran importancia para lograr la transferencia de conocimiento, fácil mantenimiento y actualización de los sistemas. Con este paso se cubre una de las alfas (**α = Documentación**)

5.- Repositorio: Finalmente el contar con un lugar donde se pueda almacenar y tener acceso a la información y documentación de los sistemas, facilita que el grupo de desarrollo cuente de manera uniforme con los elementos necesarios para poder dar mantenimiento, actualizar y resolver contingencias de manera más organizada y fluida, así como tener un mejor control de versiones. Con este paso se cubre una de las alfas (**α = Repositorio y manejo de versiones**).

Capítulo 6

Estandarizar las herramientas a emplear en la mejora de procesos

6.1 Estandarizar el documento de la toma de requerimientos

La toma de requerimientos está entre las tareas más difíciles y determinantes para el éxito de cualquier proyecto de software. Por esta razón, la mejora en la toma de requerimientos dentro de la Coordinación de Sistemas de Cómputo del IINGEN se acordó llevarla a cabo mediante la elaboración de dos tipos de documentos adaptados a las necesidades del instituto:

- Documentación de toma de requerimientos para proyectos de alto impacto. Ver **apéndice**.
- Documentación de toma de requerimientos para proyectos de bajo impacto, mantenimiento o nuevas necesidades en proyectos existentes. Ver **apéndice**.

6.1.1 Toma de requerimientos para proyectos de alto impacto

La toma de requerimientos en proyectos de alto impacto se decidió realizarla mediante un documento basado en el estándar internacional ANSI/IEEE 830 denominado SRS (Especificación de Requerimientos de Sistemas de Información) adaptado a las necesidades del instituto. En la **tabla 4** se listan los datos a incluirse en el documento y la **figura 21** muestra el diagrama de actividades del mismo.

Datos obligatorios	Datos opcionales
Nombre del sistema.	Introducción: Podrían agregarse algunas referencias necesarias para el SRS.
Siglas del sistema.	Requerimientos: Si existieran requisitos futuros, se definen en esta sección.
Introducción: En esta sección se define el propósito y el alcance del SRS.	Anexos: En esta sección se presentan algunos diagramas de apoyo y archivos necesarios para el SRS.
Descripción global: En esta sección se define el objetivo del producto y los roles de usuarios.	
Requerimientos: En esta sección se definen los requerimientos funcionales y no funcionales así como los acuerdos establecidos en los requerimientos.	
Directorio	

Tabla 4. Datos en documento de toma de requerimientos para proyectos de alto impacto.

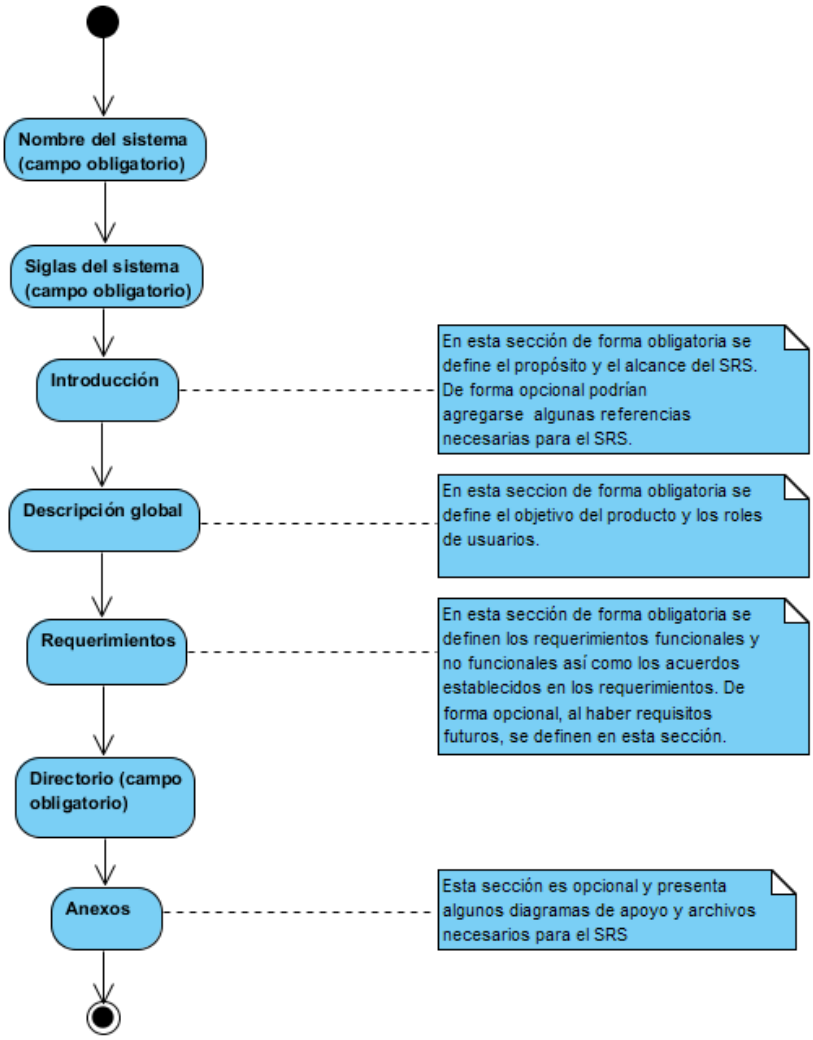


Figura 21. Diagrama de actividades del documento de toma de requerimientos para proyectos de alto impacto

6.1.2 Toma de requerimientos para proyectos de bajo impacto, mantenimiento o nuevas necesidades en proyectos existentes

La toma de este tipo de requerimientos se decidió realizarla mediante un documento parecido al empleado en algunas metodologías ágiles denominado historias de usuario (ver **capítulo 3**). En la **tabla 5** se listan los datos a incluirse en el documento y en la **figura 22.1** y **22.2** se muestra el diagrama de actividades del mismo.

Datos obligatorios	Datos opcionales
Título de la historia de usuario.	Puntos estimados: Este campo debe llenarse con el número total de requerimientos considerados para la historia de usuario.
Número de la historia de usuario.	Puntos reales: Este campo debe llenarse con el número de requerimientos autorizados y validados desde el punto de vista técnico dentro de la historia de usuario.
Nombre de la historia de usuario.	Observaciones: Aquí se escriben los detalles que no se han considerado hasta este punto y que serán de ayuda para los requerimientos dentro de la historia de usuario.
Nombre del usuario que solicita el requerimiento.	
Número de modificación del requerimiento: Este campo debe llenarse con alguno de los siguientes valores: 0,1,2...n. • Considerar como valor inicial el 1 en caso de ser mantenimiento o cambio por nuevas necesidades. • Considerar como valor inicial el 0 cuando se trata de un nuevo proyecto pequeño.	
Prioridad en negocio: Este campo debe llenarse con alguno de los siguientes valores: Alta, Media, Baja.	
Riesgo en desarrollo: Este campo debe llenarse con alguno de los siguientes valores: Alta, Media, Baja.	
Descripción.	
Iteración asignada: En caso de que la historia de usuario forme parte de una iteración, este campo debe llenarse con el número de la iteración a la que pertenece (Considerar como valor inicial el 1). En el caso de que la historia de usuario no forme parte de una iteración, este campo debe llenarse la palabra NO.	

Tabla 5. Datos a incluirse en el documento de toma de requerimientos para proyectos de bajo impacto, mantenimiento o nuevas necesidades en proyectos existentes.

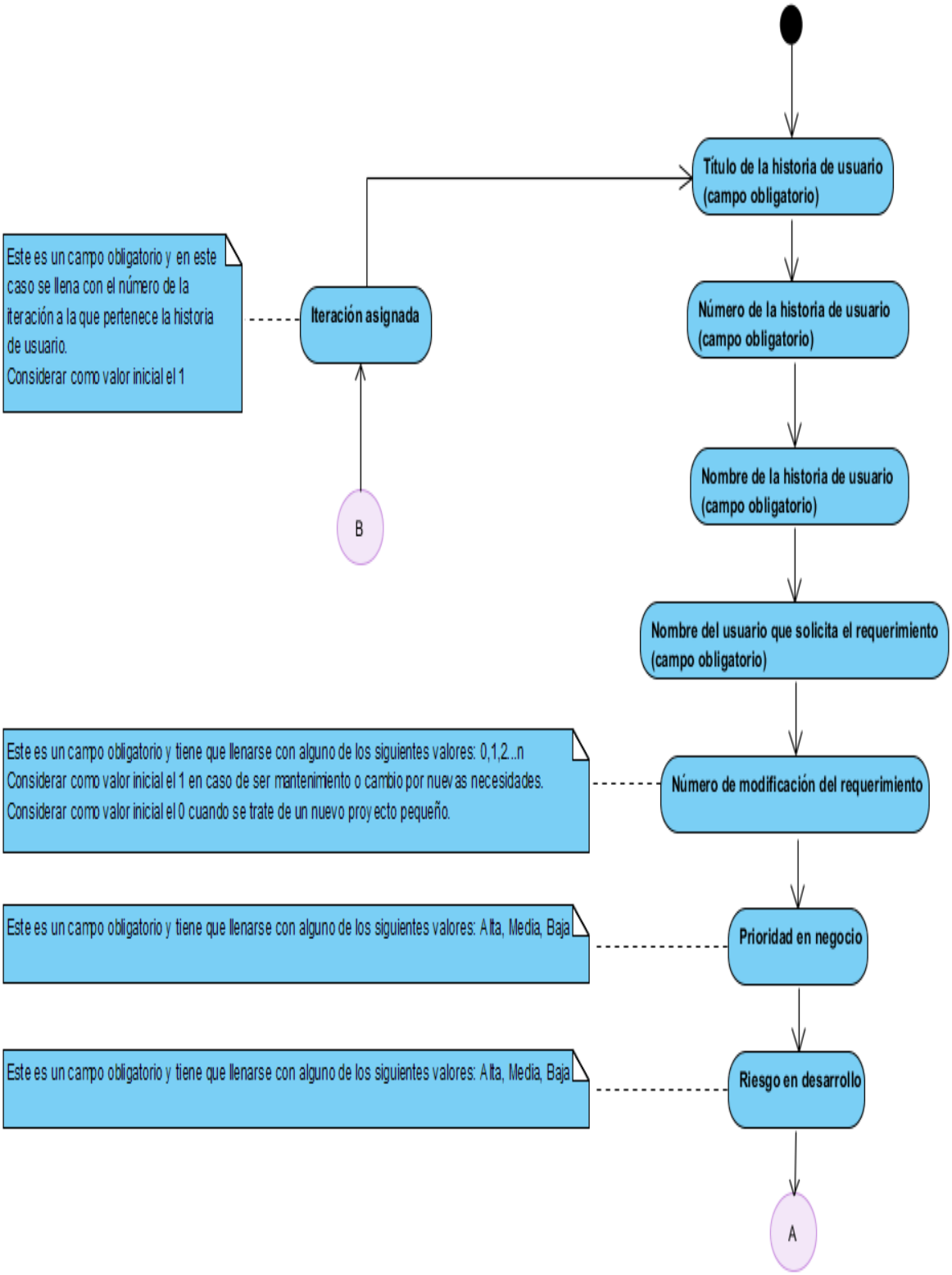


Figura 22.1 Diagrama de actividades de documento para toma de requerimientos para proyectos de bajo impacto, mantenimiento o nuevas necesidades en proyectos existentes

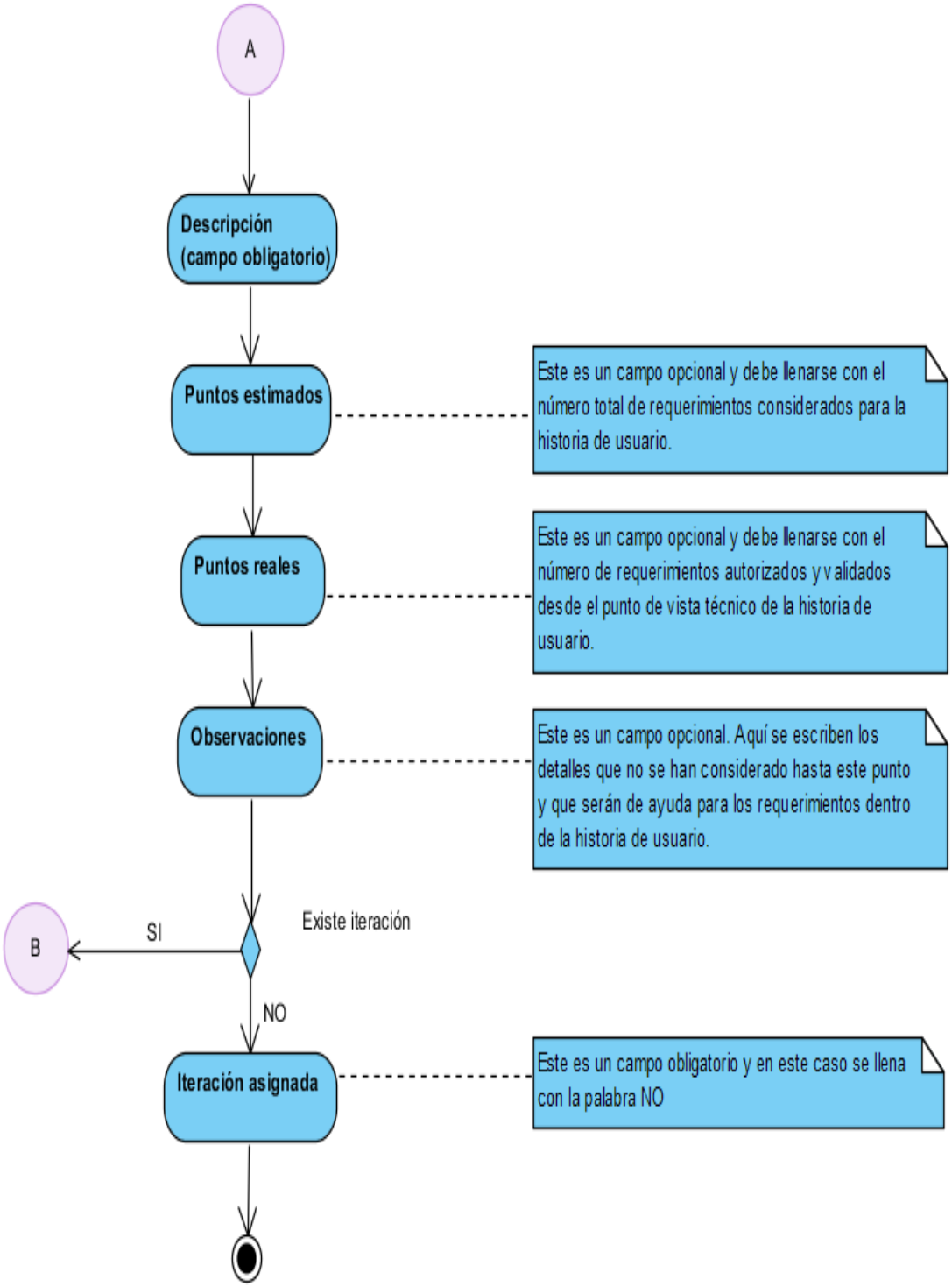


Figura 22.2 Diagrama de actividades de documento para toma de requerimientos para proyectos de bajo impacto, mantenimiento o nuevas necesidades en proyectos existentes

6.2 Normalizar estilo de código

Definir las reglas de estilo de código genera una serie de beneficios indiscutibles a los equipos de desarrollo de software:

- Mejoran la comprensión del código desarrollado por otros miembros del equipo.
- Facilitan la portabilidad y la reutilización del código desarrollado, así como su mantenimiento.
- Ayudan a la integración de nuevos desarrolladores al equipo, acelerando su integración en el flujo de trabajo.

A continuación se listan las reglas de estilo de código definidas para los lenguajes de programación utilizados dentro del IINGEN UNAM:

- Encapsulamiento: Los atributos (campos) de un objeto deben declararse privados y accederlos solamente por medio de entidades públicas como propiedades, métodos setters, métodos getters y constructores, esto con el fin de validar la asignación y obtención de valores de los atributos logrando de esta forma el control de la información que entra y sale en nuestros sistemas.
- Propiedad: Son entidades públicas creadas para poder asignar u obtener valores de los atributos privados logrando de esta forma el control de la información que entra y sale en nuestros sistemas. El nombre de una propiedad debe ser igual al nombre del atributo al que da acceso con la diferencia de que estas deben crearse con el estilo de escritura denominado upper camel case.
- Getters: Son métodos públicos creados para que solamente a través de ellos puedan obtenerse los valores de los atributos de un objeto. El nombre de estos métodos incluirá el nombre del atributo al cual accederán,

anteponiéndole al mismo la palabra get y respetando la convención utilizada por los métodos (lower camel case).

- Setters: Son métodos públicos creados para que solamente a través de ellos puedan asignarse valores a los atributos de un objeto. El nombre de estos métodos incluirá el nombre del atributo al cual accederán, anteponiéndole al mismo la palabra set y respetando la convención utilizada por los métodos (lower camel case).
- Paquetes: Los nombres de los paquetes deben estar compuestos por sustantivos en minúscula.
- Clases: Los nombres de las clases deben estar compuestos por sustantivos e incluir mayúsculas y minúsculas. Los nombres de clases deben crearse con el estilo de escritura denominado upper camel case: La primera letra siempre en mayúscula. Si el nombre de la clase está formado por varias palabras, estas deben ser separadas mediante mayúsculas.
- Interfaces: Los nombres de interfaces deben crearse con el estilo de escritura denominado upper camel case como en el caso de los nombres de las clases pero anteponiéndole al sustantivo la letra mayúscula “I”.
- Métodos: Todos los nombres de métodos deben estar compuestos por verbos o por verbos con sustantivos e incluir mayúsculas y minúsculas. Los nombres de métodos deben crearse con el estilo de escritura denominado lower camel case: La primera letra siempre en minúscula. Si el nombre del método está formado por varias palabras, estas deben ser separadas mediante mayúsculas. Para denominar a los métodos, se debe limitar el uso de caracteres de subrayado.
- Atributos (campos): Los nombres de los atributos deben estar compuestos por sustantivos y declararse como privados. Los atributos siguen la misma convención de las variables.
- Variables: Todos los nombres de variables pueden incluir letras mayúsculas y minúsculas, dígitos e incluir los caracteres de $ y “_” (debe limitarse el uso de dichos caracteres) y no deben contener espacios en blanco. Los nombres de variables deben comenzar con una letra, se permite el uso de los

caracteres $ y “_” al inicio. Los caracteres siguientes pueden ser letras, dígitos y los caracteres $ y “_”. Los nombres de variables deben crearse con el estilo de escritura denominado lower camel case: La primera letra siempre en minúscula. Si la variable está formada por varias palabras, estas deben ser separadas mediante mayúsculas. Las variables deben ser evidentes e indicar claramente su propósito a cualquier lector. Se debe evitar el uso de nombres de un solo carácter, excepto para variables temporales trowaway, tales como i, j o k, usadas como variables de control de bucle. La longitud de los nombres de variables se recomienda que sea de hasta 8 caracteres.

- Constantes: Las constantes primitivas deben escribirse en mayúsculas, con signos de subrayado como separadores de palabras. Las constantes de objeto pueden usar mayúsculas y minúsculas.
- Constructor: Son estructuras similares a los métodos, que se invocan automáticamente cuando se crea una instancia de objeto. Se suelen utilizar para inicializar con valores los atributos de un objeto. Los constructores llevan el mismo nombre de la clase que inicializará sus atributos por lo que a pesar de que son parecidos a los métodos, al nombrarse, no siguen su misma convención (lower camel case) sino que siguen la convención de las clases (upper camel case).
- Comentarios: El código debe comentarse en su justa medida, intentando que el código esté autodocumentado. Para ello se deben elegir adecuadamente los nombres de las variables, métodos y clases. El código autodocumentado es más fiable que los comentarios ya que éstos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse sólo aquello que no va a variar, por ejemplo, el objetivo de una clase o la funcionalidad de un método.

Todas las reglas establecidas anteriormente tendrán su representación gráfica en diagramas de clase como se muestra en la **figura 23**.

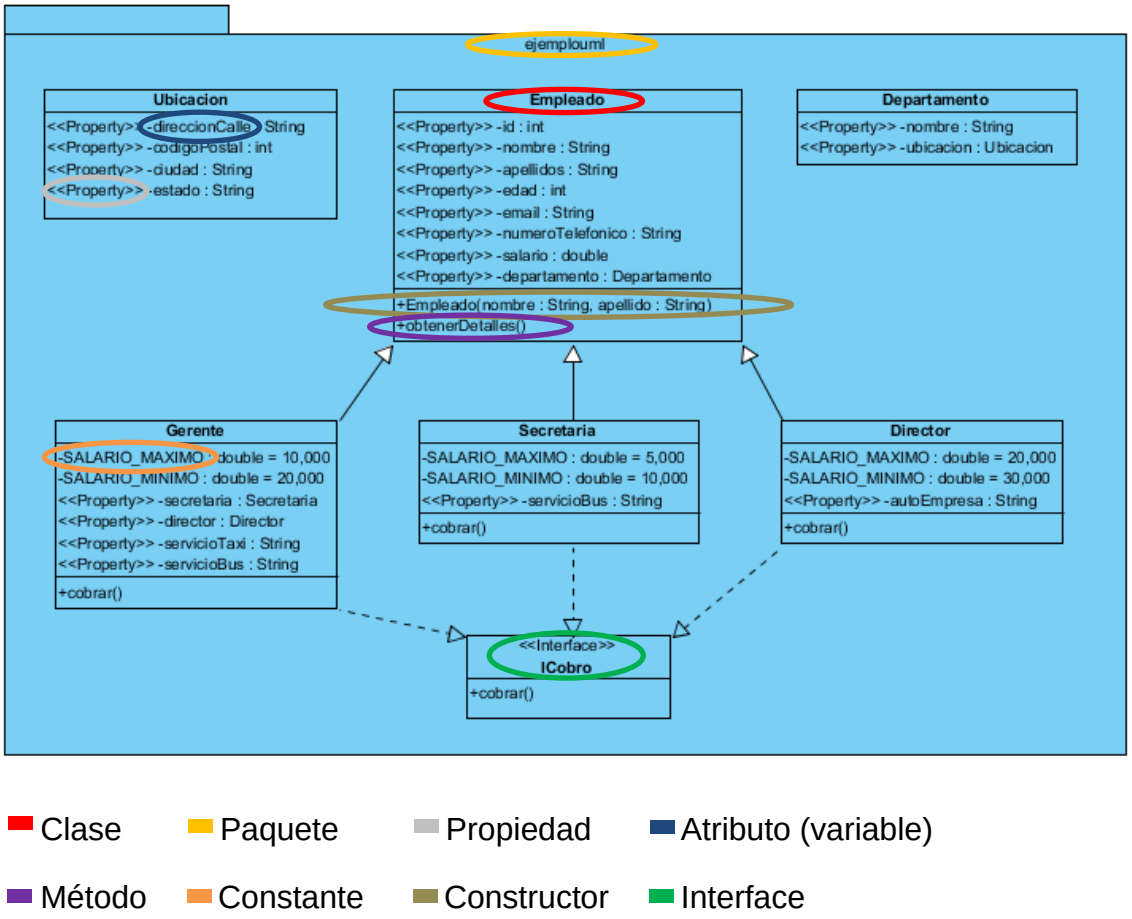


Figura 23. Ejemplo de diagrama de clase

Como se puede apreciar en la **figura 23** las entidades por las cuales se tendrá el control de los atributos de un objeto estarán representadas gráficamente por el estereotipo <<Property>> que será el equivalente de los métodos de acceso setter y getter para acceder a los atributos para los cuales fueron definidos. También se puede observar que el tipo de dato string será escrito siempre con la primera letra en mayúscula (String).

A continuación se muestra el código en lenguaje Java y C# de los elementos señalados en el diagrama de clases de la **figura 23**:

Paquetes en C#:

```
namespace ejemplouml {…}
```

Paquetes en java:

```
package ejemplouml;
```

Propiedades en C#:

```
public String Estado {
        get {
                return estado;
        }
        set {
                estado = value;
        }
}
```

Propiedades en java:

```
public String getEstado() {
        return this.estado;
}
public void setEstado(String estado) {
        this.estado = estado;
}
```

Clases en C#:

public class Empleado {…}

Clases en java:

public class Empleado {…}

Atributos en C#:

private String direccionCalle;

Atributos en java:

private String direccionCalle;

Métodos en C#:

public void obtenerDetalles() {

Métodos en java:

public void obtenerDetalles() {…}

Constantes en C#:

private double SALARIO_MAXIMO = 10,000;

Constantes en java:

private double SALARIO_MAXIMO = 10,000;

Consructores en C#:

public Empleado(ref String nombre, ref String apellido) {…}

Constructores en java:

public Empleado(String nombre, String apellido) {…}

Interfaces en C#:

public interface ICobro {…}

Interfaces en java:

public interface ICobro {…}

A continuación se listan las reglas definidas para las bases de datos creadas dentro del IINGEN UNAM:

En la definición de bases de datos se seguirá la siguiente convención:

Al definir el nombre de una nueva base de datos se incluirá al principio del mismo un indicador de la dependencia para la cual se construirá la base de datos. Este indicador será escrito en mayúsculas, seguido de un guion bajo y el nombre de la base de datos que deberá definirse con el estilo de escritura denominado upper camel case. A continuación se listan los indicadores posibles:

Secretaría Académica	ACAD_NombreBaseDeDatos
Computo	CMP_NombreBaseDeDatos
Dirección	DIR_NombreBaseDeDatos
Secretaría administrativa	ADM_NombreBaseDeDatos

Al definir el nombre de una tabla se incluirá al principio de la misma un indicador de las siglas del sistema para el cual fue creada dicha tabla. Este indicador será escrito en mayúsculas, seguido de un guion bajo y el nombre de la tabla que deberá definirse con el estilo de escritura denominado upper camel case.

Ejemplo: SIGLA_NombreTabla

Existen 8 tipos de operaciones para cualquier motor de bases de datos. Al utilizarlas en los store procedure se seguirá la siguiente convención para cada una de ellas:

- Alter sp_alter_NombreStoredProcedure
- Drop sp_drop_NombreStoredProcedure
- Delete sp_delete_NombreStoredProcedure
- Truncate sp_truncate_NombreStoredProcedure
- Select sp_select_NombreStoredProcedure
- Update sp_update_NombreStoredProcedure
- Insert sp_insert_NombreStoredProcedure

6.3 Definir documentación de valor para el desarrollo de sistemas de software

Considerando el ciclo de vida general del software, se planteó la estructura de desglose de trabajo para los proyectos creados dentro de la Coordinación de Sistemas de Cómputo del Instituto de Ingeniería de la UNAM (ver **figura 24)**.

Figura 24. Estructura de desglose de trabajo

- Requerimientos: En el proceso de toma de requerimientos, son varias las estrategias a seguir para poder obtener las necesidades de los interesados (stakeholders). El contar con un documento de especificación de requerimientos (SRS) es fundamental para una buena administración de proyectos de software, ya que al tener un SRS se delimita el alcance de los proyectos y se registran los acuerdos previos, lo que en un futuro nos da la posibilidad de utilizarlo como aval de lo que se estipuló en un principio y garantiza que el proyecto llegue a buen término y con alcances controlados. De acuerdo al proyecto o a la dinámica de los interesados, pueden emplearse dos tipos de documentos:
  - ✓ Documento de toma de requerimientos para proyectos de alto impacto.
  - ✓ Documento de toma de requerimientos para proyectos de bajo impacto. Este se utilizará en el caso de cambios o implementación de funcionalidades a los sistemas ya existentes.
- Análisis: Al igual que RUP en el IINGEN se ha optado por adoptar los diagramas de casos de uso, como una herramienta de modelado de la funcionalidad e interacción del sistema a desarrollar ya que es una técnica

eficiente para especificar el comportamiento de un sistema. Adicional a lo anterior, los diagramas de secuencia son una herramienta complementaria de los casos de uso para poder visualizar la interacción entre los casos y sus peticiones.

- Diseño: Para los sistemas basados en el paradigma orientado a objetos, se emplearán los diagramas de clases, los cuales deben ser derivados de los diagramas de secuencia. Si el sistema a desarrollar emplea un repositorio de datos relacional, entonces se empleará el diagrama entidad relación para modelar la base de datos. Para las bases de datos no relacionales, podría emplearse el diagrama de despliegue por ejemplo.
- Construcción: Para la parte del código en la construcción del sistema, se debe emplear la normalización de estilo de código como buena práctica de programación y de acuerdo a lo comentado en el apartado **6.2**.
- Pruebas: Para la parte de pruebas será necesario construir y estandarizar una serie de matrices de unidad y de integración, así como una de sistema, sin embargo, debido a que las pruebas en la mayoría de los proyectos de desarrollo de software son consideradas como algo costoso y que por cuestiones de tiempos de entrega regularmente no se implementa, consideramos en el IINGEN que es conveniente tratar este punto en una segunda iteración del proceso de mejora. Se debe trabajar con mayor profundidad en este aspecto para poder buscar estrategias que propicien la implementación de las pruebas como una herramienta de calidad y que no se consideren como una limitante para la pronta liberación de los proyectos.

6.4 Definir repositorio de código y documentación

Finalmente para las primeras cinco etapas del desglose de trabajo (**figura 24)**, se ha propuesto la utilización de herramientas que permiten tener cierta documentación de valor para la organización con el objetivo de lograr las siguientes ventajas:

- Explicar los detalles del sistema.
- Minimizar la dependencia del programador con el sistema.
- Tener mayor control para la gestión de cambios.
- Facilidad para dimensionar el impacto de los cambios o actualizaciones sugeridas en los sistemas.
- Mejorar la gestión de riesgos.

Es indispensable definir un repositorio común para el almacenamiento de los documentos y del código fuente de los sistemas de software al cual todos los miembros del equipo que participan en los proyectos puedan acceder y disponer de la información almacenada en tiempo real. La información que conviene mantener es el código fuente del aplicativo en sus diferentes versiones, los scripts de las bases de datos y cualquier elemento de interés como son documentos de análisis, diseño, construcción, pruebas y depuración, entre otros.

Un repositorio puede ser desde una simple carpeta con su estructura jerárquica, hasta herramientas de software que cuentan con la automatización de llevar un registro de cambios de quien los efectuó e incluso la posibilidad de regresar a versiones previas consideradas como estables.

Los repositorios de código y documentación elegidos para llevar a cabo el control de la información dentro de la Coordinación de Sistemas de Cómputo de la UNAM son los siguientes:

- Disco duro externo como una nube interna.
- Team foundation server.

6.5 Referencias:

Referencias web:

[15] IEEE Std. 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

[16] A New Traceable Software Requirements Specification Based on IEE 830

[17] http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

Capítulo 7

Revisar plan de mejora en su primera etapa

7.1 Revisión de metas

Al haber definido las cuatro alfas (α's):

1. Documentación.
2. Normalizar código.
3. Repositorio y manejo de versiones.
4. Requerimientos.

Nos permitió identificar las áreas de oportunidad (objetivos de interés) que tiene el IINGEN en cuanto al desarrollo de software, sobre todo nos brindó la pauta para generar una estrategia de mejora y así visualizar los primeros pasos a seguir para poder trabajar bajo el marco de referencia de ITIL, al menos en sus principios básicos que son la alineación de procesos, reducción de riesgos y garantizar el servicio.

Para poder evaluar nuestro plan de mejora propuesto, consideramos nuevamente los scripts de evaluación vistos previamente en el apartado 5.2 de este trabajo:

- Toma de requerimientos.
- Análisis.
- Diseño.
- Construcción.
- Pruebas.
- Mantenimiento.

Lo que revisaremos en los siguientes apartados es lo que estaremos logrando una vez que sea implementada la metodología de cinco pasos que se propuso para los proyectos de software.

7.2 Alineación de procesos

Toma de requerimientos

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Se cuenta con documentos estandarizados de requerimientos?	Si	2.37
2	¿Se cuenta con control y seguimiento de acuerdos?	Si	2.37
3	¿Existe una guía definida para la toma de requerimientos?	No	0
4	¿Se modelan los procesos?	Si	2.37
5	¿Se administran los requerimientos?	Si	2.37
6	¿Se cuenta con un repositorio de requerimientos?	Si	2.37
7	¿Se realizan Casos de uso de negocio?	Si	2.37

Total: 14.22 %

Análisis

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Existen documentos de Arquitectura?	No	0
2	¿Existen Casos de uso de sistema?	Si	4.15
3	¿Existe documentación estandarizada?	Si	4.15
4	¿Se cuenta con repositorio de documentos de análisis?	Si	4.15

Total: 12.45 %

Diseño

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Existen Diagramas de clases?	Si	4.15
2	¿Existen Diagramas de modelado de BD?	Si	4.15
3	¿Existe documentación estandarizada?	Si	4.15
4	¿Se cuenta con repositorio de documentos de Diseño?	Si	4.15

Total: 16.6 %

7.3 Reducción de riesgos

Construcción

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Existe Normalización de código?	Si	5.53
2	¿Existe administración de integración?	No	0
3	¿Se cuenta con repositorio de código fuente y control de versiones?	Si	5.53

Total: 11.06 %

Pruebas

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Existe especificación de casos de prueba?	No	0
2	¿Existe estandarización de pruebas?	No	0
3	¿Se cuenta con repositorio para casos de prueba?	Si	5.53

Total: 5.53 %

7.4 Garantizar el servicio

Mantenimiento

P. No.	Pregunta	Confirmación	Ideal 16.6%
1	¿Se administran los cambios y actualizaciones?	No	0
2	¿Se administran las incidencias?	No	0
3	¿Se tiene control de versiones?	No	0
4	¿Se cuenta con notas técnicas?	Si	4.15

Total: 4.15 %

De acuerdo a los resultados obtenidos de los scripts de evaluación, podemos ver una comparación de gráficos, del antes (ver **figura 25**) vs el ahora (ver **figura 26**).

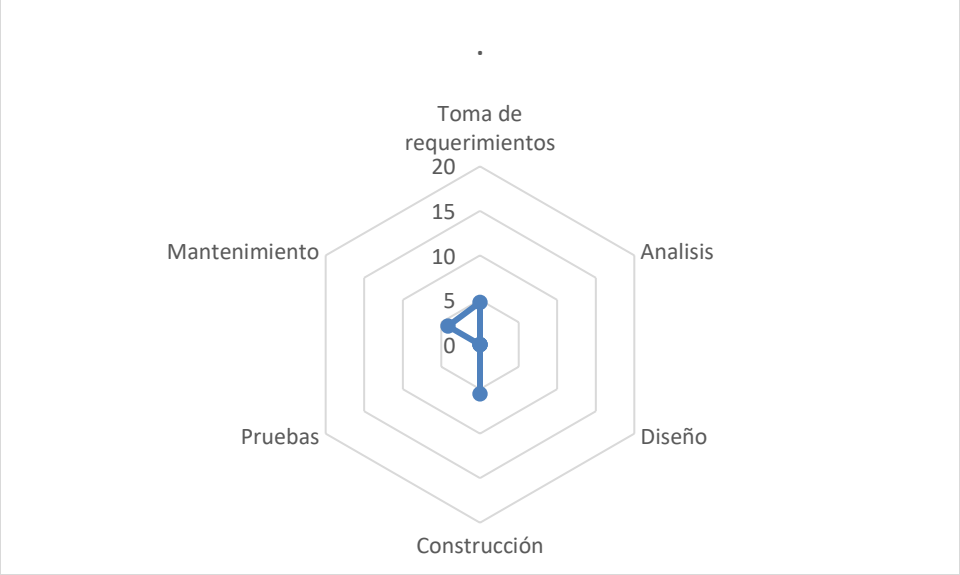


Figura 25. Practicas generales de Ingenierías de Software (antes)

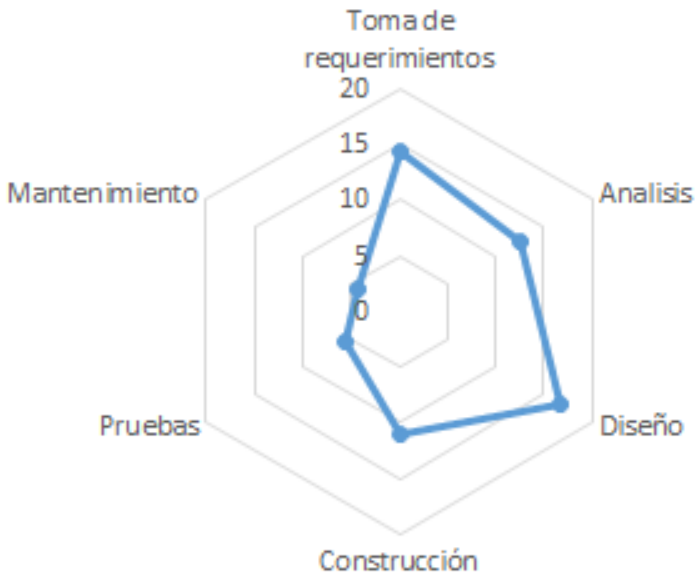


Figura 26. Practicas generales de Ingenierías de Software (ahora)

Podemos constatar que efectivamente es posible lograr una mejora considerable, que si bien es el primer esfuerzo formal por realizar un proceso más organizado al desarrollar software, sabemos que para obtener un nivel de madurez mayor es necesario continuar iterando en la metodología de cinco pasos propuesta y agregar más elementos en los scripts de evaluación.

Capítulo 8

Conclusiones

Como se observa a lo largo de este trabajo titulado “Mejora de procesos en el desarrollo de software en la Coordinación de Sistemas de Cómputo (CSC) del Instituto de Ingeniería de la UNAM (IINGEN)”, se propuso una metodología de mejora de procesos para el desarrollo de software, lo cual significó un reto pues se rompe con la inercia del solo construir software, por el de la elaboración de una serie de pasos en donde parte del proceso y clave para un mayor éxito es la generación de documentos de utilidad.

Dicha mejora de procesos, cumple con los objetivos planteados al inicio de este trabajo, pues tienen la finalidad de contar con un marco teórico que si bien no cubre a profundidad cada uno de los tópicos tratados, si muestra de manera general los estándares, modelos y herramientas de utilidad para sentar las bases teóricas y comprender lo que la Ingeniería de Software busca al ser implementada en sus diferentes niveles de madurez.

Gracias a la metodología propuesta de cinco pasos, el desarrollar software en el IINGEN está iniciando un proceso de mejora continua, que busca a través de las iteraciones empezar a generar sistemas con mayor calidad y que al contar con una metodología definida, se logre la independencia del proceso con el programador. Para lograr esto, la documentación definida y estandarizada es de gran utilidad pues de esta manera se generan las bases de conocimiento lo que propicia que cualquier miembro del equipo de desarrollo o aquellos que se integren al mismo, puedan acceder a ellas mediante los repositorios, para realizar labores de mantenimiento, actualización e inclusive una migración a versiones futuras de una manera más controlada y sobre todo organizada.

La Ingeniería de Software a pesar de ser una área de estudio que ya cuenta con grandes avances al respecto, sigue teniendo gran oportunidad de crecimiento ya que el adoptar estándares, metodologías y sobre todo buenas prácticas de documentación de los sistemas, implica un gran esfuerzo que no todas la empresas u organizaciones públicas o privadas están dispuestas a implementar inclusive por los altos costos que esto conlleva.

Debido a lo anterior y con los objetivos cubiertos, se espera también que este trabajo despierte el interés para generar nuevas estrategias que empleen ciertas metodologías que ayuden a implementar procesos de mejora aún más ambiciosos pero con menos esfuerzo y costo. Que incluyan a la ingeniería de pruebas. Que si bien en este trabajo no se tomó en consideración, no significa que sea menos importante, al contrario, es de gran valor, lo que amerita un trabajo dedicado a este ámbito. Por lo que queda abierto a nuevas ideas y metas según lo permita la mejora continua y la búsqueda de nuevos niveles de madurez.

Apéndice

Manifiesto por el desarrollo ágil de software:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Ley de Pareto

La ley de Pareto dice que cualquier conjunto de eventos que pueden asociarse a un suceso, solamente unos pocos contribuyen en forma significativa mientras que los demás son secundarios. Generalmente hay únicamente 2 o 3 causas que explican más de la mitad de las ocurrencias del suceso.

Modelo de Madurez de Capacidades (CMM)

El Modelo de Madurez de Capacidades (CMM) nació por la necesidad del departamento de defensa de los estados unidos, ya que presentaba diversos problemas con el software que encargaba desarrollar a otras empresas. Como consecuencia de estos problemas, los presupuestos se disparaban, las fechas se alargaban más y más, y en muchos casos las aplicaciones desarrolladas no correspondían con lo requerido por la organización.

Debido a que esta situación se acrecentaba más y cada vez más se volvía más intolerable, el departamento de defensa convocó un comité de expertos para que solucionasen estos tipos de problemas. En el año 1983 dicho comité concluyó que se tenía que crear un instituto de la Ingeniería del Software, dedicado exclusivamente a los problemas del software, y a ayudar al departamento de defensa.

El comité de expertos del departamento de defensa convoco entonces a un concurso público en el que expusieron las reglas de participación, en las cuales cualquiera que quisiera enviar una solicitud tendría que explicar cómo resolver diversos problemas. A esta convocatoria se presentaron diversos estamentos y la Universidad Carnegie Mellón ganó el concurso en 1985, creando el SEI (Instituto de Ingeniería del Software).

Como consecuencia, el SEI (Software Engineering Institute) se propuso crear un modelo denominado SW-CMM (CMM for software) para la evaluación de los procesos llevados a cabo por una organización.

SW-CMM es un modelo escalonado sobre el concepto de madurez que define 5 niveles o escalones para calificar la madurez de una organización. Cubre prácticas de planeación, ingeniería y administración del desarrollo y mantenimiento de software (Ver **tabla 6**).

Nivel de madurez	Áreas clave del proceso
1- Inicial Proceso caótico, impredecible. El éxito depende del esfuerzo heroico de los individuos.	Ninguna.
2- Repetible Institucionalizar procesos efectivos de administración de proyectos de software, que permiten a las organizaciones repetir prácticas exitosas desarrolladas en proyectos previos.	• Administración de requerimientos. • Planeación de proyecto de software. • Seguimiento y control del proyecto de software. • Administración de subcontratos de software. • Aseguramiento de calidad de software. • Administración de configuración de software.
3- Definido El proceso estándar para desarrollar y mantener software en la organización está documentado, incluyendo procesos de administración e ingeniería de software, y estos procesos están integrados.	• Enfoque en procesos de la organización. • Definición de procesos de la organización. • Programa de capacitación. • Administración integral de software. • Ingeniería de productos de software. • Coordinación intergrupal. • Revisiones entre colegas.
4- Administrado Se establece un conjunto de metas cuantitativas para medir el nivel de calidad y desempeño de los proyectos y del proceso organizacional.	• Administración cuantitativa de procesos. • Administración de la calidad del producto de software.
5- Optimizado No es simplemente detectar y resolver defectos, sino prevenirlos y evitarlos al implementar actividades proactivas. Mejora continua de procesos.	• Prevención de defectos. • Administración de cambio de tecnología. • Administración de cambio de procesos.

Tabla 6. SW-CMM (CMM for software)

Tras la publicación del modelo CMM for software, se comenzaron a desarrollar modelos para mejorar la madurez de las capacidades en otras áreas y ámbitos:

- P-CMM: People CMM.
- SA-CMM: Software Acquisition CMM.
- SSE-CMM: Security Systems Engineering CMM.
- T-CMM: Trusted CMM
- SE-CMM: Systems Engineering CMM.
- IPD-CMM: Integrated Product Development CMM.

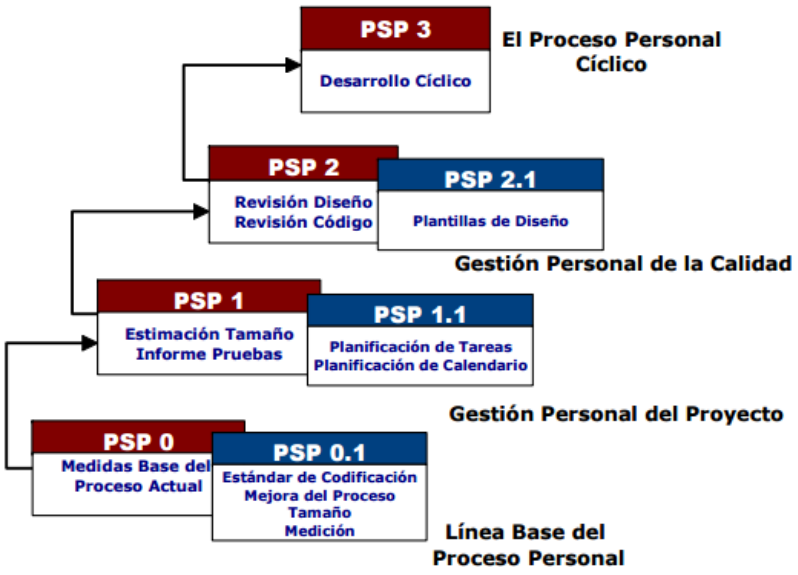
PSP – Personal Software Process

Personal Software Process (PSP) es un proceso basado en prácticas encontradas en CMM y fue diseñado para ayudar a los ingenieros de software a controlar, manejar y mejorar su trabajo. El ingeniero de software puede planear mejor el trabajo, conocer con precisión el desempeño, medir la calidad de productos, y mejorar las técnicas.

Los principios de PSP son:

- Cada ingeniero es diferente, para ser más eficiente, debe planificar su trabajo basándose en su experiencia personal.
- Usar procesos bien definidos y cuantificados
- Los ingenieros deben asumir la responsabilidad personal de la calidad de sus productos.
- Cuanto antes se detecten y corrijan los errores menos esfuerzo será necesario.
- Es más efectivo evitar los defectos que detectarlos y corregirlos.
- Trabajar bien es siempre la forma más rápida y económica de trabajar.

Incluye 7 procesos a realizar por el ingeniero de software:



PSP utiliza tres medidas base:

- Tiempo de desarrollo, defectos y tamaño.
- Todas las demás medidas son derivadas de las anteriores.

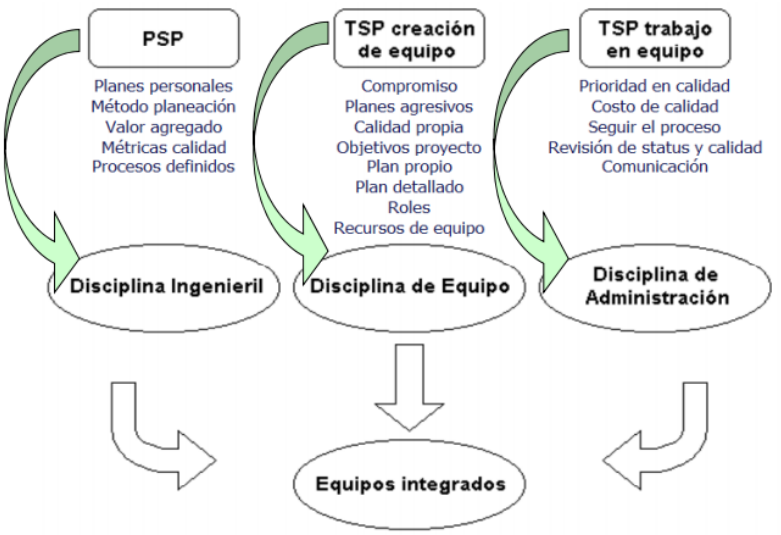
PSP puede ser aplicado en:

• Desarrollo de programas.
• Definición de requerimientos.
• Documentación.
• Pruebas de sistemas.
• Mantenimiento de sistemas.

TSP - Team Software Process

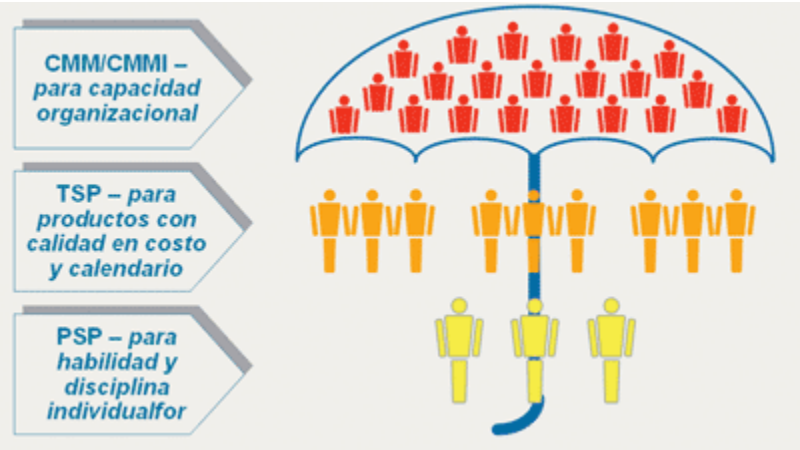
Team Software Process (TSP) es un marco para el desarrollo de software que pone igual énfasis en el proceso, producto y trabajo en equipo.

TSP se basa en PSP, y se fundamenta en que el software, en su mayoría, es desarrollado por equipos, por lo que los ingenieros de software deben primero saber controlar su trabajo, y después saber trabajar en equipo. TSP le enseña a los ingenieros a construir equipos autodirigidos y desempeñarse como un miembro efectivo del equipo. También muestra a los administradores como guiar y soportar estos equipos.



Estrategia de TSP

- Proveer un proceso sencillo basado en PSP.
- Desarrollar productos en varios ciclos. Ciclo de TSP: Lanzamiento, Estrategia, Plan, Requerimientos, Diseño, Implementación, Pruebas, Postmortem.
- Establecer medidas estándares para calidad y desempeño.
- Proveer definiciones de roles, y evaluaciones de rol y de equipo
- Requiere disciplina de proceso.
- Provee guía para manejo de problemas de trabajo en equipo.



Proceso: Un proceso es la mezcla y transformación de un grupo específico de insumos en un conjunto de rendimientos de mayor valor.

- Los insumos incluyen: Personas, materiales, equipo, información, procedimientos, políticas, tiempo, dinero.
- Los rendimientos pueden ser: La producción de un artículo, proporcionar un servicio, concluir una tarea.

La meta de cualquier proceso es transformar los insumos en rendimientos con la mayor eficacia, confiabilidad y eficiencia, así como al precio más bajo que sea posible.

- Eficacia supone calidad de un rendimiento; su influencia sobre un cliente. Un proceso eficaz satisface las necesidades de los clientes. Los rendimientos de alta calidad constituyen clientes contentos. Y éstos son buenos.
- Confiabilidad significa consistencia en el rendimiento del proceso; el nivel de calidad del rendimiento es siempre igual.
- La eficiencia se relaciona con la velocidad del proceso; cuanto tiempo es necesario para transformar los insumos en rendimientos. El tiempo del ciclo es una expresión de la eficiencia del proceso. Este es el tiempo que necesita un proceso para transformar un conjunto de insumos en rendimientos.

- La economía es el costo de transformar el conjunto de insumos en uno de rendimientos. Mientras más barato sea el proceso, mayores serán las utilidades. Muchas cosas afectan el costo del proceso. Un factor es el tiempo del ciclo.

Dentro de un proceso existe:

Trabajo: Cuando una determinada actividad hace avanzar un proceso hacia adelante o le añade valor en forma directa.

Desperdicio: Cuando una determinada actividad no hace avanzar un proceso no agrega valor al proceso. En vez de eso, solo agrega demoras y costos.

TRABAJO	DESPERDICIO
• Agrega valor • Hace avanzar el proceso	• Agrega demora • Agrega costos

El trabajo es bueno. Se desea aumentarlo al máximo. Sin embargo el desperdicio es malo. Siempre se desea eliminarlo o al menos reducirlo al mínimo.

Framework

Un framework es una estructura conceptual y un conjunto de prácticas el cual ofrece cierta funcionalidad por medio de módulos los cuales realizan labores previamente establecidas.

El objetivo de los frameworks es servir como base o soporte, sobre el cual otro proyecto de software puede ser organizado y desarrollado.

Proyectos de alto impacto

Se define como proyectos de alto impacto a aquellos que cumplen con las características siguientes:

- Cuentan con información sensible.
- Gran parte de las funcionalidades y las reglas de negocio implican un alto riesgo para la organización.

Proyectos de bajo impacto

Se define como proyectos de bajo impacto a aquellos que cumplen con las características siguientes:

- No maneja información tan sensible para el negocio.
- Las funcionalidades no tienen un gran impacto en el negocio de la organización.