ESTUDIO Y OPTIMIZACIÓN DE PROCESOS ORGANIZACIONALES DE SOFTWARE EN LA EMPRESA CEIBA SOFTWARE HOUSE S.A.S.

JHON FREDY ACOSTA DIAZ
SEBASTIAN ISAZA GOMEZ

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERIA
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
MEDELLIN
2012

ESTUDIO Y OPTIMIZACIÓN DE PROCESOS ORGANIZACIONALES DE SOFTWARE EN LA EMPRESA CEIBA SOFTWARE HOUSE S.A.S.

JHON FREDY ACOSTA DIAZ
SEBASTIAN ISAZA GOMEZ

Trabajo de grado presentado para optar al título de
Ingeniero de Sistemas

Asesor: Alberto Antonio Restrepo
Profesor Departamento de Informática y Sistemas

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERIA
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
MEDELLIN
2012

Nota de aceptación:

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Medellín, noviembre de 2012

CONTENIDO

Pág.

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

GLOSARIO

Arquitectura: La Arquitectura es el diseño de más alto nivel de la estructura de un sistema.

Atributo de proceso: Es una característica de la capacidad de un proceso que puede ser sometida a una medición.

Capacidad aumentada: Es un nivel de capacidad mayor al obtenido en el resultado de una evaluación y justificado por un plan de mejoramiento de procesos creíble.

Capacidad del proceso: Es la habilidad de un proceso para alcanzar una meta requerida.

CMMI nivel 3: Alcanzar nivel 3 significa que la forma de desarrollar proyectos está definida, es decir, está establecida, documentada y existen métricas para la consecución de objetivos concretos.

Dimensión del proceso: Es el conjunto de procesos que conforman los aspectos funcionales del modelo de referencia de procesos y la capacidad de proceso.

Evaluación de procesos: Consiste en la evaluación de los procesos de software de la organización con respecto a un modelo compatible con algún modelo de referencia.

Evaluador calificado: Es una persona que ha demostrado las habilidades, competencias y experiencias necesarias para realizar evaluaciones de procesos.

Marco teórico: conjunto de ideas o teorías que tomará el investigador para guiar su trabajo y para darle un marco ordenado y claro.

Mejoramiento de procesos: Son todas las estrategias, políticas, responsabilidades y actividades relacionadas con alcanzar las metas de mejoramiento específicas.

Optimización: Búsqueda de la mejor manera de realizar una actividad.

Propósito del proceso: Son los objetivos medibles de la ejecución del proceso y los resultados esperados de la ejecución del mismo.

Proceso: Un conjunto de actividades interrelacionadas que transforman unas entradas en salidas.

Proceso de Software: Es el proceso o el conjunto de procesos usados por una organización o proyecto para planear, gestionar, ejecutar, monitorear, controlar y mejorar sus actividades relacionadas con software

Propósito de la evaluación: Una descripción dada como parte de la entrada de la evaluación, que define la razón de realizar la evaluación de un proceso.

Reporte: El reporte puede ser la conclusión de una investigación previa o adoptar una estructura de problema-solución en base a una serie de preguntas.

INTRODUCCION

En muchísimos libros referentes a la informática y el desarrollo software, los autores hablan de que la industria de software tiene gran impacto en prácticamente todas las ramas del desarrollo de la sociedad. Sin embargo, a pesar de sus reconocidos resultados, aún resulta significativo el número de proyectos de software que no culminan con éxito.

> Entre los principales problemas se identifican: insuficiente calidad de los productos finales, estimaciones inexactas de la duración de los proyectos, retrasos en la entrega de los productos, descontrol de los costos de desarrollo y mantenimiento de los productos, pobre definición de los requisitos, descontrol de los requerimientos de cambios.
> Las principales dificultades apuntan a la no existencia de procesos de desarrollo de software bien definidos que garanticen un buen uso de los talentos y recursos con que cuentan las organizaciones y que estos procesos mejoren de forma continua. Sin embargo, para lograr las mejoras deseadas resulta vital entrenar al personal de forma tal que sea capaz de desempeñar los roles que les corresponda de manera disciplinada ya sea trabajando individualmente o en equipo. La universidad debe jugar un papel protagónico en la formación del ingeniero informático en correspondencia con las exigencias que demanda la industria. Por su parte, la industria se siente presionada por mejorar su desempeño y desarrollar software de calidad, sobre todo teniendo en cuenta que en la actualidad se experimenta un crecimiento del volumen y complejidad de los productos de software que, indudablemente, ha convertido el desarrollo de sistemas en una actividad de equipo (Watts, 1998).

Es de aclarar que la tecnología cambia rápidamente, y es un reto para las empresas dedicadas a este sector estar al tanto de estos cambios, adicionalmente estas empresas deben de estar cambiando constantemente sus procesos de acuerdo con los cambios en las tecnologías, con el fin de hacer sus procesos más eficientes y eficaces.

La mejora de procesos involucra el uso de nuevas herramientas, técnicas, recopilación de información del proceso en cuestión, los productos y las personas.

Ceiba Software House es una empresa del sector informático, que se encuentra en el reto de mejorar de forma continua sus procesos organizacionales con el fin de garantizar un nivel de madurez óptimo para las necesidades que exige el mercado nacional. Para lograr esto, se necesita la implementación de modelos y normas.

El presente trabajo pretende mostrar el estado actual de la empresa con referente a ciertos procesos que fueron seleccionados en el core del negocio.

Según el estado actual que se encuentre, se evaluará la posibilidad de mejorar el proceso basados en los casos de éxito que se encuentran vigentes en el mercado.

De los modelos que vamos a considerar, se tiene RUP, AUM, SCRUM, PSP, TSP, ISO IEC 15504 entre otros. Ellos nos darán un marco teórico para dar sugerencias de cómo alternativamente se puede mejorar el proceso.

Es de aclarar que la norma ISO 15504 nos define como debe de estar construido y que documentación se debe tomar en cuenta en el momento de crear procesos organizacionales, ya que estos se pueden volver factores críticos a la hora de capacitar a los nuevos integrantes de los procesos o si uno de los integrantes se retira llevándose su conocimiento.

A su vez es de importancia que los procesos sigan una normativa y sean estandarizados ya que estos nos definen como estamos versus la competencia, el nivel de satisfacción de los clientes, el nivel de calidad que se está manejando.

1. METODOLOGÍA

Nota: *Por petición estricta de parte de la compañía CEIBA Software House S.A.S, el capítulo referente al estado actual de esta, no será presentado en este trabajo por tratarse de información confidencial. A partir de ahora, CEIBA será llamada como ‘’la compañía’’.*

1.1 SET DE INFORMACIÓN

La información tomada, ha sido recopilada de una serie de revistas científicas, periódicos y trabajos de investigación enfocados a la mejora de procesos, selección de roles y metodologías de trabajo en equipo y desarrollo de software.

1.2 BÚSQUEDA DE INFORMACIÓN

La información tomada del set de datos, fue adaptada de acuerdo con las necesidades encontradas en la compañía. Es de aclarar que fue necesario combinar conceptos y metodologías con el fin de buscar las mejores alternativas que se adapten a las necesidades del negocio.

1.3 SELECCIÓN DEL SET DE DATOS

La selección del set de datos consistió en cuatro etapas las cuales fueron:

Primero, la selección de artículos basados en el título. En este nivel, los elementos que no eran de relevancia para el estudio fueron descartados, pero, dado el caso de encontrar dudas con respecto a la información que estos contenían, se mantuvieron hasta que se analizará la información del mismo.

La segunda etapa consistió en la selección de los artículos basados en el abstract y las introducciones. En esta etapa, nos enfocamos primordialmente en los casos de éxito donde las metodologías generaron mejoras considerables sobre los procesos organizacionales.

La tercera etapa consistió en la selección de artículos basados en los gráficos, tablas con recopilación de datos y conclusiones respecto a las metodologías adoptadas en el proceso de mejora, y los retos encontrados durante el mismo.

Finalmente en la cuarta etapa nos concentramos en la selección de información teórica donde se explican de forma técnica las metodologías, conceptos y normas como base para abordar los estudios realizados.

2. OBJETIVOS

2.1 OBJETIVO GENERAL

Plantear alternativas de mejoras según el estado actual en que se encuentran los procesos de desarrollo de la compañía.

2.2 OBJETIVOS ESPECÍFICOS

- Mostrar el estado actual de la compañía en el proceso de desarrollo.
- Investigación de metodologías para el desarrollo de software, casos de buenas prácticas en el mercado.
- Estándares que rigen el mercado actual en la mejora de procesos.
- Se realizará una investigación retrospectiva con el fin de identificar cual es la mejor alternativa dadas ciertas características de proyectos software.

3. ¿QUE ES UN PROCESO?

Un proceso se define como:

> Conjunto de actividades que se interrelacionan y que interactúan entre sí, transformando un elemento de entrada en un elemento de salida.
> Todos los procesos deben de estar alineados con los objetivos de la organización y deben estar diseñados para agregar valor.
> Cada proceso cuenta con clientes internos y externos que pueden estar afectados por el proceso y son finalmente los que definen el elemento de salida que requieren, de acuerdo a sus necesidades y expectativas (Velásquez, 2012).

3.1 LA IMPORTANCIA DE LOS PROCESOS EN UNA ORGANIZACIÓN

Una buena definición y un claro entendimiento de los procesos que se utilizan en una compañía son una base importante para el buen funcionamiento de esta y un claro camino hacia el cumplimiento de las metas. Nos puede ayudar a resolver problemas graves, ya que conoceremos las preguntas básicas del qué hacemos, cómo lo hacemos y cuánto tardamos de forma muy realista.

La definición de los procesos es una de las herramientas esenciales más importantes para la mejora continua dentro de la organización.

Los procesos se utilizan para ‘’entender y/o perfeccionar los procesos existentes y para diseñar nuevos procesos’’ (Velásquez, 2012).

La retroalimentación ‘’permite asegurarse de que los procesos están correctamente diseñados, así como detectar las carencias y necesidades de los clientes’’ (Velásquez, 2012).

Contribuyen también a definir influencias en otros procesos para ayudar al equipo a entender mejor la complejidad.

Implementar procesos en una empresa ofrece gran soporte al modelo de negocios de cada organización. Las organizaciones han puesto en marcha como prioridad la administración de procesos como un marco de referencia para la realización, coordinación y retroalimentación de sus actividades diarias. Este marco, ha ayudado a replantear y hacer más eficiente el uso y manejo de recursos en las empresas y con su implementación, una cadena de valor mejor soportada, mayor claridad en las actividades que corresponden a un rol, incremento en la calidad de producto y mayor satisfacción del cliente.

4. LA IMPORTANCIA DE LA EVALUACIÓN DE LOS PROCESOS

‘’Lo ideal para una organización sería considerar su rendimiento frente a la competencia. El rendimiento debe estar relacionado con los procesos (y tecnología) que son usados para determinar cómo los procesos contribuyen al éxito del negocio’’ (Rational Software, 2011). Como no siempre es posible obtener información del rendimiento de la competencia, y en especial de sus procesos, las organizaciones necesitan un sistema de medición independiente para sus procesos.

Modelos como la norma ISO/IEC 15504 y CMMI nivel 3 proveen un marco de medida para la evaluación de procesos que son reconocidos a nivel internacional.

Durante los tiempos, las empresas han reconocido que operar con base en procesos o mejora de los mismos no es fácil. Implica reconocer si la organización tiene:

- Los conocimientos técnicos, herramientas y métodos adecuados para ejecutarlos.
- El apoyo y autoridad de los directivos para incentivar la pronta implementación.
- Considerar la posible automatización de un proceso ya existente si es necesario.
- Evaluar si hay apertura para cambiar el enfoque del sistema de medición del desempeño

Sin embargo vale la pena considerar este reto. "La alineación a procesos es importante porque cuando se diseñan, se basan y alinean a los objetivos de cada organización, y a su modelos de negocio, garantiza que las empresas dejan de ser

departamentos independientes y se transforman en sistemas integrados, a eso nos llevan los procesos" (Reyes, 2009).

Para lograr en la organización la mejora de procesos y obtención de beneficios se deberán de considerar los siguientes aspectos:

- Plan de mejora continua, que consiste en planear, hacer revisar y mejorar.
- Elementos del diseño del proceso, consiste en poseer las herramientas y documentar el proceso.
- Conocer las necesidades de los clientes.
- Personal entrenado con las capacidades para ejecutar las actividades del proceso.
- Metodología de sustento del proceso, es recomendable contar con un marco de referencia sobre los procesos y un sistema de apoyo en sus interrelaciones con otros procesos.
- Marco de nivel de madurez, es decir, usar un marco de referencia para identificar cómo los procesos van adquiriendo madurez.

La evaluación de procesos debe tener un conjunto de características que la hagan útil para la organización. Estas características, son:

- Un marco de trabajo que asegure consistencia en la evaluación.
- Un marco de medición que especifique la escala de rendimiento o de “performance”.
- Una descripción de los procesos que la organización desea evaluar.

- Un método que asegure la consistencia de la evaluación hecha por los evaluadores.
- Una guía de cómo adaptar la evaluación de procesos para que le sirva a la organización.
- Requisitos de evaluadores que aseguren que tienen las capacidades para efectuar las evaluaciones.
- Guía que explique cómo aplicar los resultados de la evaluación

‘’Para alcanzar el éxito se necesita una hoja de ruta que sirva de guía para la organización y que además deje como evidencia el estado actual en el cual se encuentren los procesos y el conjunto de acciones que deben ser tomadas para que los procesos cumplan las expectativas que la organización requiere’’(Ocampo y Ruiz, 2008).

La evaluación de procesos ayudaría a alcanzar lo anteriormente mencionado:

- Determinando el estado actual de los procesos organizacionales, recolectando evidencia objetiva con respecto a los modelos de evaluación de procesos.
- Suministrando una guía a las organizaciones en la selección de los procesos a implementar, según modelos de referencia de procesos que definan los procesos.
- Suministrando guía a las organizaciones para ayudar en la mejora de sus procesos a través de escalas de medida de procesos.

La idea principal de la evaluación de procesos consiste en comparar un proceso implementado real con un conjunto de características de procesos estándar (atributos) que hacen parte de descripciones de procesos y de un marco de medición.

Para evaluar un proceso se utilizan los siguientes pasos:

- La evaluación de procesos califica si la realización de los procesos utiliza todas las actividades básicas (Prácticas Base) necesarias para alcanzar los objetivos.

- La evaluación de procesos califica qué tan bien (o mal) un proceso es gestionado comparando la realización, administración, definición, medida y mejoramiento de los procesos con una escala estándar de buenas prácticas administrativas.

- Se elige un modelo de referencia de procesos que suministre un marco coherente para los procesos en el cual la organización pueda escoger si implementa o mejora para alcanzar los resultados deseados. Estos modelos pueden cubrir desarrollo de software, ingeniería de software, recursos humanos u otras áreas de la organización.

En definitiva, el conocimiento y la mejora de los procesos de la compañía puede ser muy importante para la mejor gestión y aprovechamiento, también, para hacernos crecer y a tener controlada nuestra capacidad empresarial.

4.1 CONSIDERACIONES PARA MEJORAR UN PROCESO

Las empresas tienen que definir qué procesos deben mejorarse. Es importante seleccionarlos atendiendo a problemas reales o potenciales evidenciados. Por ejemplo:

- Problemas y/o quejas con clientes externos e internos
- Procesos con altos costos
- Procesos con tiempo de ciclo prolongado
- Incorporación de nuevas tecnologías
- Perdidas de mercados
- Existencia de peleas o malas comunicaciones interfuncionales.
- Visualización de mejoras al realizar la representación del proceso.
- No se está cumpliendo con las especificaciones establecidas.

4.2 APLICACIÓN DE LA OPTIMIZACIÓN DE UN PROCESO

‘’Las organizaciones pueden mejorar los tres aspectos fundamentales en sus operaciones, mejorando los tres actores que intervienen en esas operaciones, que son las personas, los procesos y los productos. Cada uno de estos actores tiene sus beneficios y limitaciones en términos de mejoramiento, pero en general es importante mejorar los tres’’ (Ocampo y Ruiz, 2008).

Optimizar un proceso debe conducir a la mejora del producto y también debe proveer el medio para la mejora de las personas que intervienen en ese proceso y

producto. La optimización puede darse con mejoramiento continuo o a través de innovación.

Una buena planeación en el programa de mejoramiento de procesos debe valerse tanto de mejoramiento continuo como de innovación. Esto permitirá crear sinergias en las personas, procesos y productos, lo que conduciría a la organización a ser competitiva y exitosa.

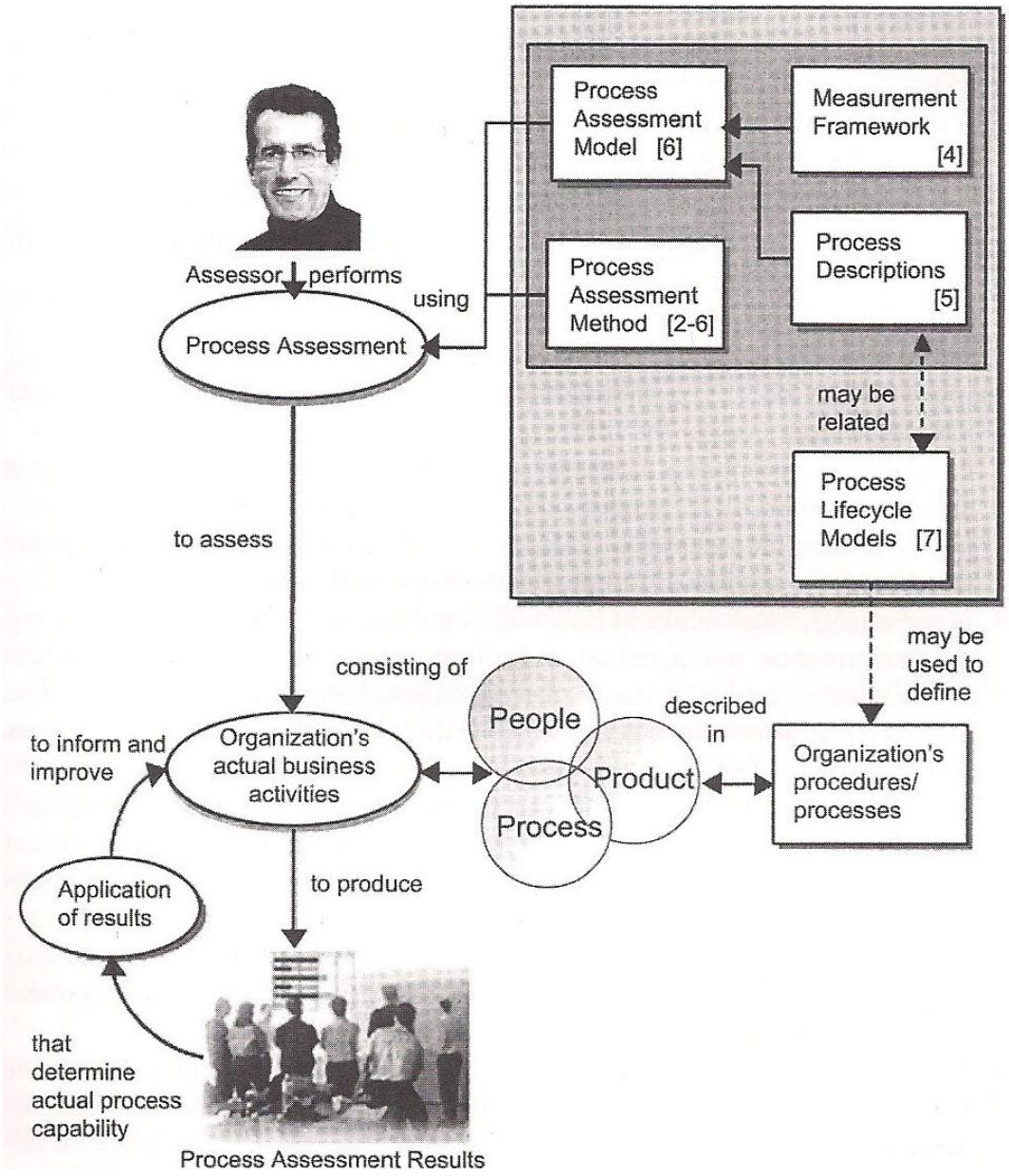
Es muy importante que el proceso de optimización apunte a:

> Alcanzar los objetivos del negocio (efectividad) con la menor cantidad de recursos posibles (eficiencia) y la menor posibilidad de fracaso (riesgo). Para crear de manera eficiente productos y servicios nuevos y mejorados, las organizaciones están recurriendo cada vez más a procesos automatizados.
> Los recursos disponibles para el mejoramiento de procesos por lo general son muy limitados en las organizaciones. Es por esto que se debe invertir cuando el retorno de la inversión sea maximizado (Ocampo y Ruiz, 2008).
>
> Para hacer una eficiente optimización de los procesos, la organización debe usar una escala de medición clara y consistente. La escala de medición debe hacer una clara distinción entre los niveles logrados. Estos niveles no solo ayudan a la organización a entender su situación actual en términos de rendimiento, si no también sirve como guía para saber qué hacer en el futuro.
>
> Una simple analogía de esta escala puede compararse con una casa:
>
> NIVEL 0: Es como vivir al aire libre debajo de un árbol. No es bueno
> NIVEL 1: Es como tener un piso, cuatro paredes y un techo.
> NIVEL 2: Se tiene paredes y ventanas, ya se tiene control del entorno de tu casa
> NIVEL 3: Se tiene algunos servicios básicos en la casa como electricidad, ya se tiene un mejor control
> NIVEL 4: Se tiene todo los muebles necesarios, se siente ya como estar en casa y pensar en qué tipo de comodidad puedo desear
> NIVEL 5: Ya se tiene todo lo deseado en cuestión de comodidad, seguridad, lo último en entretenimiento, rápida conexión a internet (Van Loon, 2007).

En adición de una buena escala de medida, la optimización de procesos requiere un constante acercamiento.

El siguiente diagrama ilustra algunas de las actividades y documentos que se necesitan para estar dirigido a asegurar una optimización consistente.

Figura 1. Visión general de la optimización de procesos



Fuente: Van Loon, 2007

Como se muestra en la figura 1, la optimización de procesos mejora las actividades actuales de la organización en términos de uno o más procesos implementados contra un framework de medida.

5. INTRODUCCION A LA NORMA ISO/IEC 15504

‘’El ISO/IEC 15504, también conocido como *Software Process Improvement Capability Determination*, abreviado SPICE, en español, «Determinación de la Capacidad de Mejora del Proceso de Software» es un modelo para la mejora y evaluación de los procesos de desarrollo y mantenimiento de sistemas de información y productos de software’’ (Wikipedia, 2012).

5.1 HISTORIA

La ISO (International Organization for Standardization) y la IEC (International Electrotechnical Committee) crean en 1987 el Joint Technical CommittEe (JTC), cuyo objetivo es elaborar estándares para las tecnologías de la información. Este Comité se compone de 19 Subcomités, cada uno relacionado con distintos aspectos de las tecnologías de la información.
El Subcomité 7 (SC7) es el encargado de la Ingeniería del Software y está dividido en 9 grupos de trabajo (WG). Por ejemplo, WG2 System Software Documentation, WG4 Tools and Environment, etc. El objetivo de este Comité es la estandarización de procesos, herramientas y tecnologías de apoyo para la ingeniería de productos de software y de sistemas.
El grupo de trabajo WG10 se encarga de desarrollar estándares y guías que cubren métodos, prácticas y aplicaciones de valoración de procesos de adquisición, desarrollo, entrega, operación, evolución y servicios de productos software (Ocampo y Ruiz, 2008)

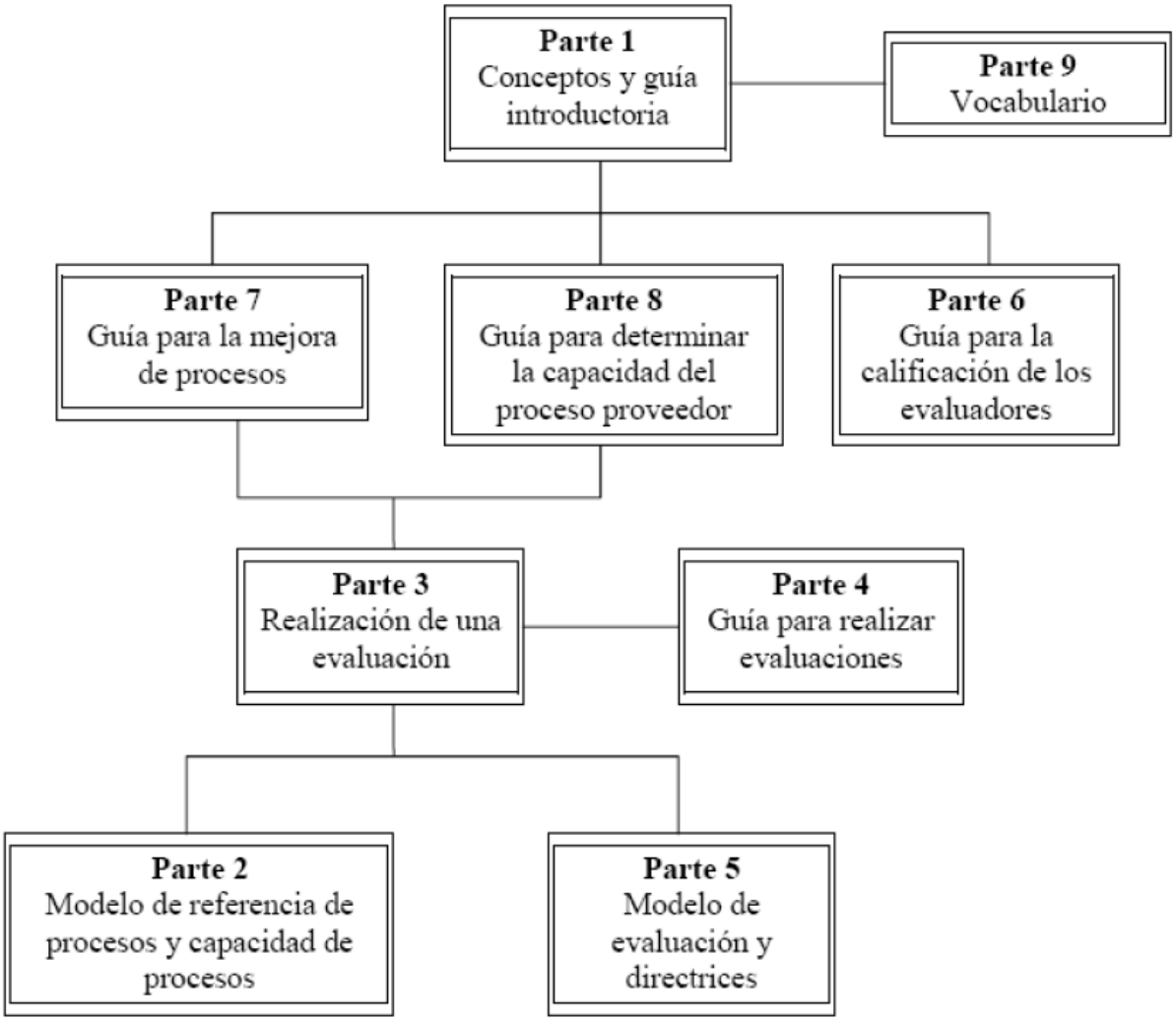
Se creó entonces el proyecto SPICE, que es una importante iniciativa internacional que hasta hace poco existía en apoyo de la Norma Internacional ISO/ IEC 15504 para Evaluación de Procesos (software). Por tanto, el proyecto SPICE fue creado bajo los auspicios del Comité Internacional de estándares de Ingeniería de Software y Sistemas a través de su Grupo de Trabajo sobre Evaluación de proceso (WG10).
En 1992, el informe del grupo de estudio dijo que: “...la comunidad internacional debería poner recursos para desarrollar un estándar para la evaluación de procesos software, incorporando lo mejor de los métodos de evaluación de procesos existentes (Wikipedia, 2012).

En 1998 se publica la primera versión del estándar como Informe Técnico (en 1995 se publica como „borrador“), evolucionando posteriormente hacia Estándar Internacional, con la realización de tres fases de pruebas.
La Fase 1 (1995), con la idea de validar la decisiones de diseño y usabilidad del borrador; la Fase 2 (1996-1998) que a los objetivos anteriores sumaba

proveer de una guía de aplicación y revisar la consistencia, validez, adecuación, usabilidad y portabilidad de SPICE.
La Fase 3 (hasta marzo de 2003, en que se cierra el proyecto SPICE) se realiza con la idea de aportar entradas y publicar el estándar ISO. Tras los ensayos, comienza la fase de Benchmarking (actual fase), con la idea de recolectar datos de los procesos de evaluación y analizarlos y comienza la publicación de partes del estándar (Ocampo y Ruiz, 2008).

La primera versión del estándar publicada como Informe Técnico, consta de las partes que se muestran en la Figura 2.

Figura 2. Informe técnico: Componentes y las relaciones entre ellos



Fuente: Wikipedia, 2012

El marco proporciona un enfoque estructurado de la evaluación de los procesos software, mediante el cual:
- Se anima a la auto-evaluación.
- Se aborda la idoneidad de la gestión de los procesos evaluados
- Tiene en cuenta el contexto en el que operan los procesos evaluados
- Produce un conjunto de valores del proceso (perfil del proceso) en vez de un resultado (válido/no válido)
Este marco es válido para todos los dominios de aplicaciones y tamaños de organización (Ocampo y Ruiz, 2008).

5.2 CARACTERISTICAS

Establece un marco y los requisitos para cualquier proceso de evaluación de procesos y proporciona requisitos para los modelos de evaluación de los procesos.
Proporciona también requisitos para cualquier modelo de evaluación de organizaciones.
Proporciona guías para la definición de las competencias de un evaluador de procesos.
Actualmente tiene 10 partes: de la 1 a la 7 completas y de la 8 a la 10 en fase de desarrollo.
Comprende: evaluación de procesos, mejora de procesos, determinación de capacidad.
Proporciona, en su parte 5, un Modelo de evaluación de procesos para los procesos de ciclo de vida del software definidos en el estándar ISO/IEC 12207 que define los procesos del ciclo de vida del desarrollo, mantenimiento y operación de los sistemas de software.
Proporciona, en su parte 6, un Modelo de evaluación de procesos para los procesos de ciclo de vida del sistema definidos en el estándar ISO/IEC 15288 que define los procesos del ciclo de vida del desarrollo, mantenimiento y operación de sistemas.
Proporcionará, en su parte 8, un Modelo de evaluación de procesos para los procesos de servicios TIC que serán definidos en el estándar ISO/IEC 20000-4 que definirá los procesos contenidos en la norma ISO/IEC 20000-1 (Wikipedia, 2012).

5.3 DIMENSIONES

Tiene una arquitectura basada en dos dimensiones: de proceso y de capacidad de proceso. Define que todo modelo de evaluación de procesos debe definir: - la dimensión de procesos: el modelo de procesos de referencia (dimensión de las abscisas) - la dimensión de la capacidad: niveles de capacidad y atributos de los procesos. Los niveles de capacidad para todo modelo de evaluación de procesos pueden tener desde el 0 y al menos hasta el nivel 1 de los siguientes niveles de capacidad estándar:

- Nivel 0: Incompleto
- Nivel 1: Realizado
- Nivel 2: Gestionado
- Nivel 3: Establecido
- Nivel 4: Predecible
- Nivel 5: En optimización

Para cada nivel existen unos atributos de procesos estándar que ayudan a evaluar los niveles de capacidad (Wikipedia, 2012).

Descripción de los niveles de capacidad:

5.3.1 Nivel 0. Incompleto

El nivel 0 representa un proceso que no se ha ejecutado o que no alcanza su propósito. Se asume que un proceso que está siendo evaluado es necesario y que por lo tanto tiene un propósito. Este nivel de capacidad puede aparentar ser superfluo, pero de hecho, la cantidad de actividad puede variar de no realizado completamente a realizado, pero sin alcanzar el propósito del proceso.
Cuando un proceso tiene un nivel 0 quiere decir que no existe evidencia sistemática (productos de trabajo) que demuestre que el proceso está implementado. Los productos de trabajo son artefactos asociados con la ejecución de un proceso. Las organizaciones que poseen procesos en este nivel tienen problemas para entender el aspecto fundamental del propósito del proceso y por lo tanto implementan una actividad para alcanzar este aspecto (Ocampo y Ruiz, 2008).

5.3.2 Nivel 1. Realizado

El nivel de capacidad 1 indica que el proceso alcanza su propósito.

5.3.3 Nivel 2. Gestionado

Este nivel especifica que el proceso está gestionado, es decir, que existe planeación, monitoreo, y ajustes para el proceso. Además se requiere que los productos de trabajo sean establecidos, controlados y que estén bajo mantenimiento.
En el nivel 2, la organización administra el rendimiento del proceso. Este es el primer nivel de capacidad en el cual se hace explícita la gestión de proceso y además el rendimiento del proceso es ajustado de ser necesario, en caso de no alcanzar las necesidades y los objetivos propuestos.
La gestión de los procesos está enfocada en asegurar que la organización conozca qué produce (productos de trabajo), para cuándo necesita ser

producido (plan, programa), que los resultados sean acordes con las necesidades organizacionales y que el proceso logre el rendimiento planeado de manera confiable. La gestión de proceso tendrá como resultado unos artefactos y/o actividades que serán verificables (Ocampo y Ruiz, 2008).

5.3.4 Nivel 3. Establecido

‘’Este nivel especifica que el proceso gestionado hace uso de un proceso definido. Este proceso definido está basado en un proceso estándar que es adaptado según las necesidades y desplegado a las unidades de negocio pertinentes’’ (Ocampo y Ruiz, 2008).

5.3.5 Nivel 4. Predecible

‘’Este nivel especifica que el proceso establecido opera dentro de límites definidos. Un proceso predecible usa mediciones cuantitativas y utiliza técnicas administrativas para operar el proceso dentro de los límites definidos’’ (Ocampo y Ruiz, 2008).

5.3.6 Nivel 5. En optimización

Este nivel de capacidad especifica que el proceso predecible (en nivel de capacidad 4) es mejorado de forma continua para alcanzar las metas proyectadas del negocio.

5.4 OPTIMIZACIÓN DE PROCESOS Y LA NORMA ISO/IEC 15504

La optimización de procesos puede utilizar una gran variedad de enfoques para lograr todos los requerimientos y necesidades que hemos hablado. Diferentes modelos han sido desarrollados durante los tiempos, muchos de ellos tienen incompatibilidades con otros.
ISO/IEC 15504 es un estándar que provee un camino claro para lograr todas esas necesidades específicas y/o se refiere a todos los requerimientos y necesidades para optimizar procesos. Muchas organizaciones han abrazado este estándar. Van desde los estándares internacional y nacionales hasta organizaciones profesionales como lo es la “Software engineering institute” en

EEUU, “The European software institute” y “Software quality institute in Australia’’ (Wikipedia, 2012)

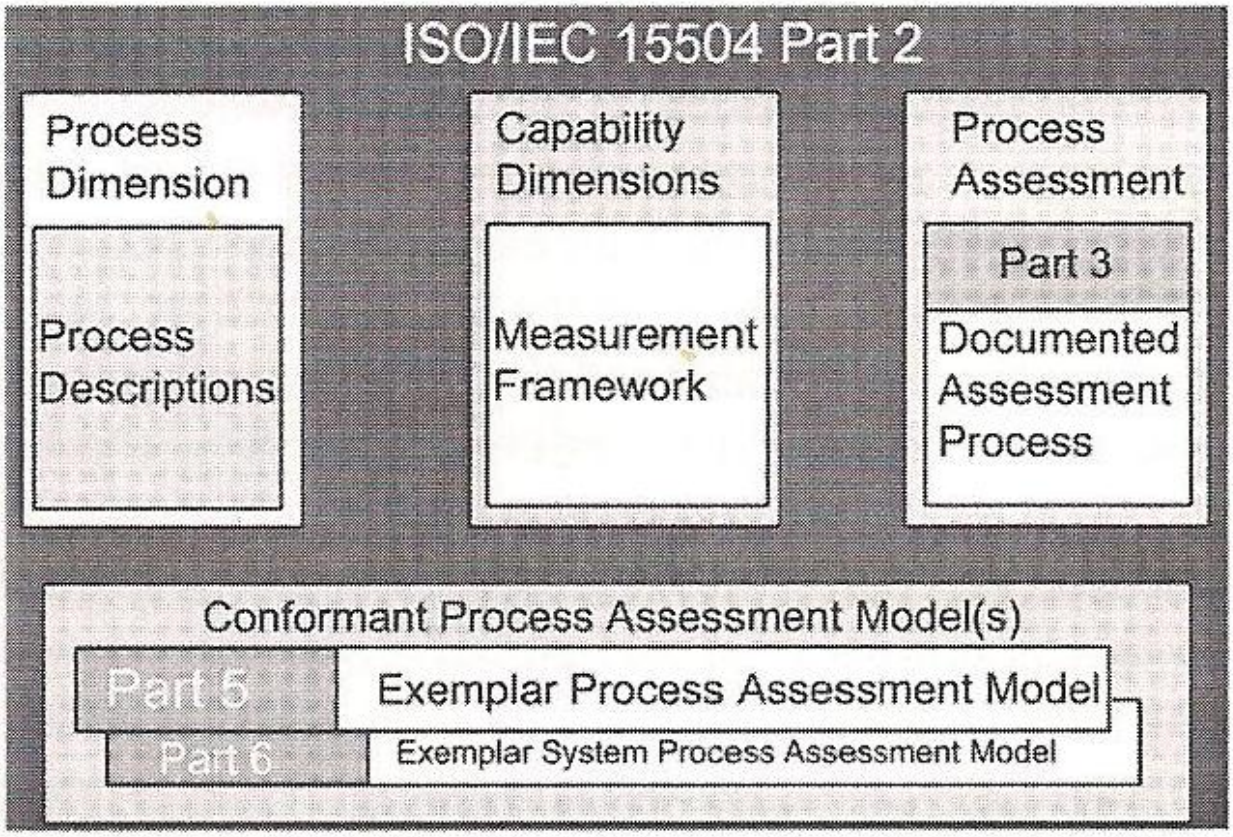
Las empresas usan el estándar como un medio para:

- Evaluar la capacidad de los empleados.
- Proveer un medio para asegurar que los procesos consistentes estén aplicados de una manera que produzca productos de alta calidad.
- Ayudar a los empleados a ser mejores en la administración de los procesos.

Uno de los aspectos fuertes de la norma ISO/IEC 15504 es que este define cómo se encuentra la organización y provee una dirección clara de cómo mejorar los procesos. La capacidad de mejorar procesos puede ayudar al negocio a mejorar su eficiencia y efectividad.

El siguiente diagrama ilustra todos los requerimientos básicos para la optimización de procesos.

Figura 3. Requisitos básicos para la optimización de procesos



Fuente: Van Loon, 2007

ISO/IEC 15504 ‘’part 2’ es ayudado por el ‘’part 3’’, que provee guías informativas para aplicarlos a la optimización de procesos, ‘’y por el part 5 y 6” que provee modelos ejemplares.

5.5 REQUERIMIENTOS DE LA EVALUACIÓN DE UN PROCESO

ISO/IEC 15504 define los requerimientos para la evaluación de un proceso. Esos requerimientos se describen a continuación:

- Usar un proceso de evaluación que esté documentado, el cual debe permitir alcanzar el propósito de la evaluación.

- Definir roles y responsabilidades del sponsor y de los evaluadores, tanto calificados como no calificados.
- Definir las entradas para cada evaluación.
- Conducir la evaluación, de acuerdo con el proceso de evaluación documentado.
- Registrar las salidas de la evaluación según el estándar.

Se requiere una serie de pasos para efectuar exitosamente una evaluación de procesos:

- Planeación
- Recolección de datos
- Validación de datos
- Calificación de atributos de proceso
- Reporte

5.5.1 Planeación

La planeación depende principalmente del propósito de la evaluación y del alcance. El propósito y el alcance influencian los procesos a ser evaluados, las actividades que deberán tomarse, los recursos requeridos (personas, lugares y herramientas), presupuesto, y el cronograma propuesto.

Una evaluación consume recursos de la organización, por lo tanto es importante que en la planeación se defina claramente los roles y responsabilidades para los

participantes principales, con el fin de ayudar a maximizar la eficiencia y la efectividad del proceso de evaluación.

La mayoría de métodos de evaluación dependen de las entrevistas a personas que ejecutan el proceso, por lo tanto es muy importante seleccionar cuidadosamente a quién se va a entrevistar y preparar la(s) entrevistas de tal manera que las personas puedan describir los procesos y proveer datos que ilustran la implementación del proceso.
Además de esto, durante la planeación, las entradas y las salidas de la evaluación necesitan estar indicadas y acordadas. Esto puede incluir documentos de proyecto, descripciones de proceso de las unidades organizacionales y datos de implementación de procesos. Las salidas deben incluir reportes preliminares y finales, registros de las sesiones de discusión planeadas sobre los resultados, incluyendo participantes, Sponsor y administración (Ocampo y Ruiz, 2008).

5.5.2 Recolección de datos

Antes de empezar con la evaluación dentro de la organización, existe una necesidad de recolectar datos sobre la unidad organizacional, sus procesos y aquellos datos en el modelo de evaluación, usados para evaluar esos procesos. Por lo tanto, el equipo de evaluación necesita establecer una correspondencia entre los procesos del negocio y el modelo.
Durante la evaluación, los evaluadores necesitan recolectar evidencia que apoye el alcance de la evaluación y su propósito. Los evaluadores necesitan recolectar datos por cada proceso evaluado. Esos datos pueden ser recolectados de diferentes formas, entre las que se incluyen entrevistas, cuestionarios, discusiones generales, y revisiones de documentos (Ocampo y Ruiz, 2008).

5.5.3 Validación de datos

Esta actividad se relaciona con revisar los datos recolectados para ver si son representativos de los procesos evaluados y si además cumplen con el propósito y el alcance de la evaluación. Además, los evaluadores necesitan revisar que los datos recolectados a través de los procesos son consistentes.

Los casos en los cuales la evaluación haya sido una repetición para confirmar mejoramiento de procesos, el evaluador puede comparar los datos anteriores contra aquellos recolectados para confirmar los mejoramientos.

Los evaluadores deben usar la presentación de resultados preliminares a la organización como otra base para validar que los datos representan la unidad organizacional evaluada. Cuando los resultados preliminares están en disputa, los evaluadores necesitan nuevos datos o datos adicionales para validar sus

descubrimientos. Si esto resulta ser imposible, entonces los evaluadores necesitarán plantear claramente en los reportes de evaluación junto con un análisis, de lo que significa en términos de calificación, oportunidades de mejoramiento y riesgos de mal interpretación de resultados (Ocampo y Ruiz, 2008).

5.5.4 Reporte

El reporte se puede enfocar en dos tipos de detalle:

- Determinación de la capacidad y de riesgos asociados a los espacios en la capacidad objetivo.
- Oportunidades de mejoramiento.

Normalmente los evaluadores deben hacer una presentación preliminar de los resultados como parte de la validación de los datos, para asegurarse de que el reporte final refleje el funcionamiento de la unidad organizacional evaluada, cualquier retroalimentación de la unidad, y además proveer una forma clara y fácil de interpretar el resultado.

El reporte final también debe reflejar el propósito de la evaluación (por ejemplo, el mejoramiento y/o enfoque en la determinación de la capacidad), junto con información adicional acordada (por ejemplo, oportunidades de mejoramiento, planes de acción, benchmarks, comparación con resultados de evaluaciones previas o contra otras unidades organizacionales).

La introducción de la determinación de un nivel de madurez organizacional resultará en una extensión de la determinación de la capacidad. En este caso, las determinaciones de capacidad serán agregadas dentro del nivel de madurez organizacional (Ocampo y Ruiz, 2008).

5.6 OBJETIVOS DE SPICE

El objetivo del modelo SPICE es proporcionar un marco de referencia para la valoración de los procesos software, fomentar la calidad de los productos software y generar un proceso de valoración repetible comparable y verificable.

Este marco de referencia puede ser usado por las empresas proveedoras de software para:
- Comprender el estado de sus propios procesos para la mejora de los mismos.
- Determinar la idoneidad de sus propios procesos para un requerimiento particular.
- Determinar su capacidad ante un determinado contrato.

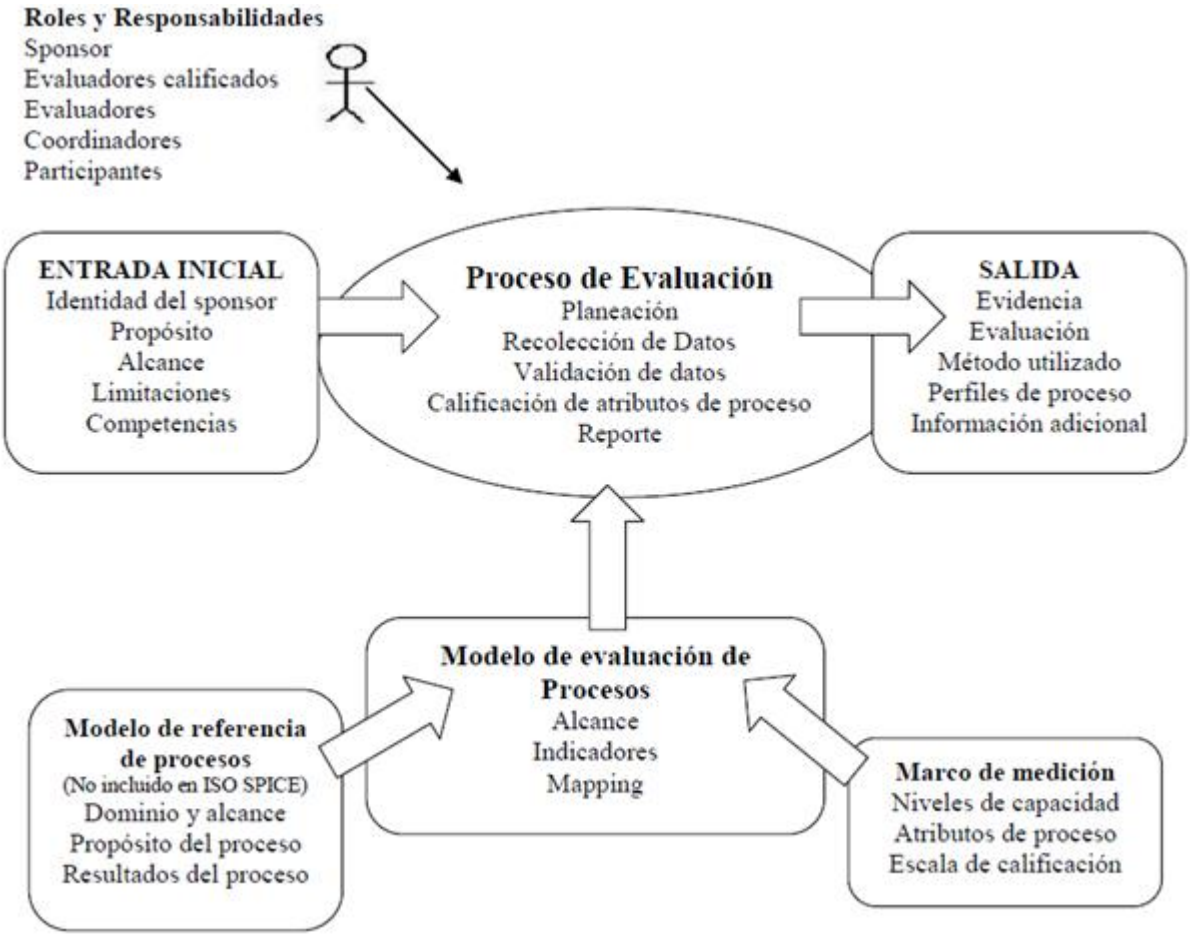
- Determinar la conveniencia de procesos de otras organizaciones para un contrato particular (Ocampo y Ruiz, 2008).

5.7 ESTRUCTURA DE SPICE

Los requerimientos de ISO/IEC 15504 especifican una relación entre un modelo de referencia de procesos, un modelo de evaluación de procesos y un proceso de evaluación. Como el estándar consiste en esencia en la evaluación de procesos, desde un punto de vista práctico, las organizaciones y los evaluadores están interesados principalmente en cómo hacer evaluaciones adecuadas. El estándar requiere cinco elementos para ser definidos:

- Un proceso de evaluación adecuado
- Un modelo de evaluación de procesos adecuado.
- Roles y Responsabilidades asignados, con especial énfasis en evaluadores competentes (o calificados).
- Entradas de evaluación específicas
- Salidas de evaluación específicas.

Figura 4. Modelo de estructura de SPICE



Fuente: Van Loon, 2007

Como se muestra en la figura 4, el estándar requiere un modelo de evaluación de procesos que se construye a partir de un marco de medición y un modelo de referencia de procesos que no hace parte del estándar. El modelo de medición provee los niveles de capacidad, los atributos del proceso y la escala de calificación. El modelo de referencia provee dominio, alcance, y el propósito y resultado del proceso. Este modelo de referencia normalmente se toma a partir de una norma llamada ISO/IEC 12207, que presenta facilidad de trabajo con ISO/IEC 15504, aunque no necesariamente se debe trabajar con esa norma, pues existen una serie de pasos que permiten usar otros modelos de referencia existentes que

perfectamente permiten utilizarse junto con SPICE, según las necesidades de la organización.

5.8 CALIFICACIÓN NPLF PARA LOS ATRIBUTOS DE PROCESO

N – No ejecutado. No hay evidencia o hay muy poca para demostrar que el atributo ha sido ejecutado [0 a 15 %].

P – Parcialmente ejecutado. Existe evidencia de que un atributo ha sido ejecutado, pero el grado de ejecución podría ser impredecible [16 a 50%].

L – Ejecutado en gran parte. Existe evidencia significativa de la ejecución del atributo de forma sistemática [51 a 85%].

F – Ejecutado completamente. Existe evidencia de ejecución completa y sistemática del atributo [86-100%].

6. MODELOS DE REFERENCIA DE PROCESOS

Los modelos de referencia de procesos han sido diseñados para guiar a los usuarios en la definición e implementación de los procesos y no tienen nada que ver con la evaluación de procesos. Estos modelos son externos al estándar SPICE y la organización puede escoger el modelo más apropiado a sus necesidades.
Los modelos agrupan los procesos en categorías como Desarrollo, Administración y Soporte, como una guía a los usuarios acerca del potencial de cada proceso. Esto puede ser útil desde el punto de vista de una persona o equipo interesado en implementar los procesos (Ocampo y Ruiz, 2008).

6.1 MODELO DE EVALUACIÓN DE PROCESOS

ISO / IEC 15504 requiere de un Modelo de Evaluación de Proceso compatible que proporcione un nivel de detalle que garantice la coherencia de los resultados de la evaluación. El Modelo de Proceso de Referencia no puede ser utilizado por sí solo como base para realizar de forma confiable una evaluación de la capacidad de procesos, porque el nivel de detalle no es suficiente.

Por lo tanto, es necesario un Modelo de Evaluación de Proceso como base para la recolección de pruebas y calificación de la capacidad de los procesos. El Modelo de Evaluación de Procesos se basará en los principios de gestión de procesos, utilizando el criterio de que la capacidad de un proceso puede evaluarse demostrando el logro de cada uno de sus atributos de proceso. Dicho de otro modo, esto significa que una evaluación global de capacidad se puede construir a partir de la evaluación de cada uno de sus componentes (Ocampo y Ruiz, 2008).

6.2 USO DE INDICADORES

Un Modelo de Evaluación de Procesos debe documentar una serie de indicadores de la capacidad y el rendimiento de los procesos, que permita juzgar la capacidad de los procesos para ser una base sólida en las pruebas objetivo. Los indicadores de evaluación representan diversos tipos de pruebas objetivo que se deben encontrar cuando un proceso es utilizado, y por lo tanto proporcionar una base para juzgar el logro de la capacidad. Los indicadores se dividen en dos categorías, en relación con las dos dimensiones del modelo:

- Los factores que indican el rendimiento del proceso.
- Los factores que indican su capacidad.

Los Indicadores de rendimiento del proceso proporcionan cobertura a los efectos y los resultados del proceso. Los Indicadores de capacidad de proceso en el modelo se encargan de proporcionar la cobertura para los atributos del proceso en los distintos niveles de capacidad.

Los correspondientes indicadores de desempeño de procesos serán diferentes de proceso a proceso, pero por lo general consisten en:

- Identificar los productos de trabajo que entran al proceso (entrada de productos de trabajo) y sus características.
- Identificar los productos de trabajo que son producidos por el proceso (salida de productos de trabajo) y sus características.
- Actividades que transforman la entrada de productos de trabajo en salida de productos (Indicadores de Prácticas Base) e indicar el cambio de estados.

Una práctica base es un indicador que es equivalente a una actividad o tarea que aborda la finalidad de un proceso particular. Cuando la organización realiza actividades equivalentes a las prácticas base asociadas con un proceso, puede lograr constantemente el propósito del proceso. Los indicadores de prácticas base se describen a un nivel abstracto, identifican el "qué" se debe hacer sin especificar el "cómo" (que representan las actividades funcionales del proceso).

En los niveles más altos de la dimensión de proceso, los indicadores de capacidad de proceso están relacionados con los atributos de la capacidad de procesos para cada **Nivel de Capacidad.**

7. SCRUM

7.1 PROPOSITO

SCRUM no es un proceso o una técnica para desarrollar o crear productos, sino que es una metodología en la cual se pueden emplear diferentes procesos o técnicas, con el fin de aflorar la eficiencia de las diferentes prácticas de desarrollo del equipo de trabajo, a su vez proporciona un marco para desarrollar productos complejos.

7.2 TEORIA SCRUM

“Scrum, que se basa en la teoría del control empírico de procesos, emplea un enfoque iterativo e incremental para optimizar la previsibilidad y controlar los riesgos. Existen tres pilares que sostienen toda implementación del control empírico de procesos” (Schwaber y Sutherland, 2011).

Cruz emplea bloques de tiempo para crear regularidad. Los elementos de Cruz basados en bloques de tiempo son: la Reunión de Planificación de la Entrega, la Reunión de Planificación del Sprint, el Sprint, el Scrum Diario, la Revisión del Sprint, y la Retrospectiva del Sprint. El corazón de Scrum es un Sprint, que es una iteración de un mes de duración o menos. La duración de cada Sprint se mantiene constante a lo largo de todo el esfuerzo de desarrollo. Todos los Sprints utilizan el mismo marco de referencia de Scrum, y proporcionan un incremento de funcionalidad potencialmente utilizable al producto final. Cada Sprint se inicia inmediatamente después del anterior. Scrum emplea cuatro Artefactos principales.

El Product Backlog es una lista priorizada de todo lo que podría ser necesario en el producto. El Sprint Backlog es una lista de tareas para convertir el Product Backlog correspondiente a un Sprint, en un incremento del producto potencialmente entregable. Un burndown es una medida del backlog restante a través del tiempo. Un Burndown de Versión mide el Product Backlog restante durante el tiempo correspondiente a una liberación de una versión. Un Sprint Burndown mide los elementos restantes del Sprint Backlog en el transcurso de un Sprint. Las Reglas sirven de unión para los bloques de tiempo, los roles y los artefactos de Scrum, y se describen en el cuerpo de este documento.

8. METODOLOGÍAS ÁGILES VS METODOLOGÍAS FORMALES

A continuación, comenzaremos a comparar importantes metodologías del campo del software que para nuestro propósito, es darnos una visión más clara y saber qué aspectos de estas metodologías son de interés para el crecimiento organizacional, tomando en cuenta que estará basado en un proceso investigativo retrospectivo, y en las buenas prácticas que se puedan identificar en estos.

8.1 METODOLOGÍAS ÁGILES

Estas metodologías representan un cambio en el paradigma del desarrollo formal de software basado en procesos. Uno de los factores de formación que obligaron a los fundadores fueron varios estudios que demostraron altos índices de fracaso asociados a los grandes esfuerzos disciplinados de software. Metodologías ágiles hace hincapié en un rápido retorno de la inversión.

El sitio web del Manifiesto de las metodologías ágiles, describe los siguientes cuatro principios que destacan su postura en contra de la disciplina tradicional de desarrollo de software

- Individuos e interacción sobre procesos y herramientas
- Trabajo software a través de una amplia documentación
- Colaboración con el cliente sobre negociación de contratos
- Responder al cambio sobre seguir un plan

Metodologías ágiles se encuentran en contraste con CMMI y otros procesos de desarrollo en el que rechazan muchos puntos de vista tradicionales acerca de la planificación detallada y la definición de los requisitos. Este enfoque está motivado

en la dinámica de los requisitos y que los clientes no siempre saben lo que necesitan hasta que vean qué se está produciendo.

Estas Metodologías reconocen que los requisitos particularmente aquellos que involucran soluciones desarrolladas no son fáciles de imaginar. Insisten en que los procesos de desarrollo software deben ser flexibles o de lo contrario, los procesos se rompen dando lugar a clientes insatisfechos.

Representan un conjunto de principios para el desarrollo de software más que una metodología específica.

Las características de un enfoque ágil incluyen el desarrollo gradual que consta de pequeñas versiones de software con ciclos rápidos, acuerdos de colaboración con los clientes.

Abrahamsson identifica las deficiencias en las siguientes perspectivas:

> - Ciclo de vida del desarrollo software: La cobertura del ciclo de vida necesita ser explicado, y las interfaces con las fases que no están cubiertas, se deben aclarar
> - Gestión de proyectos: La armonización es necesaria para apoyar las actividades de gestión de proyectos
> - Soluciones en cualquier situación: Más trabajo sobre cómo adoptar métodos ágiles de desarrollo en situaciones que se necesite
> - Apoyo empírico: Los resultados de cualquier trabajo experimental, deben ser publicados (Abrahamsson et al, 2003).

8.2 METODOLOGÍAS FORMALES

El área de los Métodos Formales Orientados a Objetos (MFOO en adelante) se ocupa de la descripción de software de manera precisa y rigurosa. Tal objetivo obliga a la utilización de lenguajes de especificación de software de naturaleza matemática.

Históricamente, los métodos formales centraron sus objetivos en la calidad, descuidando, en gran medida, el entorno en el que debían aplicarse. La falta de educación general y el déficit de herramientas los convirtieron en recursos ideales y difíciles de manejar. La sistematización progresiva de la producción de software necesita de notaciones expresivas y, evidentemente, formales. No hay automatización sin formalización.

‘’Metodología formal es un conjunto de técnicas de modelado para especificar, desarrollar y verificar sistemas software mediante el uso del lenguaje matemático y características orientadas a objetos’’ (Mendelson, 1987).

Muchos de los métodos formales presentes en la literatura aparecen como métodos formales extendidos con conceptos de orientación a objetos. Es decir, la existencia de un lenguaje formal suele ser previa a la existencia del lenguaje orientado a objetos.

Dada la diversidad de trabajos y formalismos existentes en la actualidad, es necesario establecer qué se entiende por formal y qué se entiende por orientación a objetos.

Los métodos formales representan un área de conocimientos exigente porque obligan a amalgamar conocimientos de índole matemático deductivo, conocimientos sobre diseño de sistemas orientado a objetos y conocimientos sobre modelos de producción de software orientado a objetos. Por lo tanto, la aproximación al tema debe hacerse de forma incremental. Inicialmente se necesita introducir los principios básicos y generales sobre los conocimientos previamente aludidos. Posteriormente, se asume que los métodos formales se encuentran fuertemente influenciados por el modelo matemático subyacente y por tanto, existe la necesidad de conocer dichos modelos.

8.2.1 Conceptos básicos

Los conceptos básicos no sólo incluyen fundamentos matemáticos y sobre orientación a objetos sino también fundamentos sobre procesos de producción de software.

‘’Entre los fundamentos matemáticos básicos se encuentran las nociones de lenguaje formal, sintaxis, semántica, modelo, satisfacción, teoría y prueba. La aproximación a tales temas debería ser genérica para evitar una presentación sesgada hacia determinado formalismos’’ (Manna y Pnueli, 1991).

‘’Entre los fundamentos sobre orientación a objetos básicos se destaca el concepto de concurrencia. Diferentes modelos de concurrencia (entrelazado y ejecución solapada) y conceptos como interferencia y sincronización son fundamentales para entender la evolución de sistemas orientados a objetos en el tiempo’’ (Conboy y Fitzgerald, 2004).

Los métodos formales no se deben considerar de forma aislada al proceso de desarrollo de software. Un conocimiento básico de un proceso resulta fundamental para contextualizar adecuadamente el uso y resultados esperables para un método formal.

8.3 HACIA UN MARCO CONCEPTUAL DE LAS METODOLOGÍAS ÁGILES: UN ESTUDIO ÁGIL EN DIFERENTES DISCIPLINAS

La explicación de este marco conceptual con referente a las metodologías ágiles, fue extraída de una investigación hecha por Kieran Conboy (Conboy y Fitzgerald, 2004).

8.3.1 Deficiencias del estudio de agilidad en desarrollo software

No existe una definición universalmente aceptada sobre lo que es un método ágil en el campo de los sistemas de información y desarrollo software. Cockburn aunque descarta la existencia de un método ágil completo, afirmando que es algo que los desarrolladores sólo pueden aspirar, y sólo retrospectivamente puede determinar si un método ágil se adhirió a la realidad.

La razón de esta falta de consenso es que los principios de agilidad expresados en el manifiesto ágil (agilemanifesto.org) carecen de base en teoría de la gestión y filosofía, y no tiene en cuenta la evolución del concepto de agilidad en campos fuera del desarrollo software. Como resultado, hay muchos métodos actualmente en uso que son todos categorizados como ágil en aquellos que utilizan estos métodos. Cada uno de ellos se centra en gran medida en algunos de los principios del manifiesto e ignora otros por completo. Por lo tanto, la búsqueda de una solución definitiva, que abarca todo concepto de agilidad no se encuentra simplemente a través de un examen de agilidad en otros campos. Más bien, se encuentra a través de un examen de los conceptos subyacentes de la agilidad y flexibilidad.

8.3.2 Explicación del marco

- El objetivo general de un método ágil es identificar y manejar el cambio
- Identificar y manejar el cambio de recursos requerido. Los equipos de desarrollo se enfrentan a la tarea de hacer frente al cambio y reducir al mínimo el coste de la calidad, tiempo y la disminución de la obligación de hacerlo.
- A continuación vamos a mostrar las cuatro categorías que un equipo ‘’ágil’’ puede llevar a cabo en relación con el cambio:

- Creación de cambio: Aquí es donde el equipo es el instigador principal del cambio, sin ser pasivos y sólo sujeto a cambio que se origina en el cliente o en niveles superiores de la organización.
- Proactivo: Aquí es donde el equipo toma acciones para provocar cambios antes de que ocurran. Los prototipos son un excelente ejemplo de esto. el retraso de las decisiones de inversión y puesta en escena de los recursos son también ejemplos de pro actividad.
- Reacción: Se trata de acciones tomadas por el equipo en respuesta al cambio.
- Robustez: La robustez se caracteriza a menudo como un componente ágil. Sin embargo, este marco reconoce que la robustez no es una actividad en sí mismo, pero es un producto de la pro actividad. En otras palabras, las actividades proactivas, si se hace bien, debería reducir la necesidad de reacción. Reaccionar menos requiere un nivel mayor de robustez.

8.3.3 Medición de actividades ágiles usando el marco de evaluación

8.3.3.1 Creación de medición, pro acción y reacción

Medición de las actividades creativas, proactivas y reactivas en términos de su nivel de agilidad, se hace comparando el número de cambios identificados y realizados por una actividad con el costo de llevar a cabo esa actividad. Cuanto mayor es el número de cambios por costo, es más ágil la actividad

8.3.3.2 Medición de robustez

Robustez a diferencia de los otros aspectos ágiles no es una cifra dinámica, y no hay actividades explícitas que contribuyan a la robustez. En este marco, la robustez verdadera existe cuando la necesidad de reaccionar es inexistente. Por

lo tanto, las actividades menos reactivas en relación a un cambio, es más robusto el proceso

8.3.3.3 Medición de aprendizaje

El componente de aprendizaje en métodos ágiles no es algo que puede medirse tan fácilmente como las otras actividades. La razón de esto es que las entradas a estos componentes, es decir, coste, tiempo y defectos se pueden medir, pero la salida no puede. Esto es por que no hay cambios identificados o realizados como resultado del proceso de aprendizaje.

- **Creación (del cambio)**

El concepto de creación es sencilla en el desarrollo software. En términos simples, cualquier acción o habilidad puede ser considerado como creación de cambio si éste causa un cambio que puede no haber ocurrido antes de la acción. Lo más importante que destacar es el cambio de amplio alcance que puede tener en un entorno de desarrollo. Al hacer referencia a los controladores genéricos mencionados de cambio, un número de ejemplos de creación al cambio pueden ser extrapolados. Estos ejemplos se muestran en la siguiente tabla:

Tabla 1. Tabla que muestra ejemplo de creación de cambio

Source of Change	Type of Change	ISD Example
Customer	Changes in Demand	The ISD team may take action to encourage new requirements. For example, taking the time to select or develop a number of alternative IS solutions, apart from the single solution preferred by the client.
Competition	Increased Cost Pressures from Competitors	An ISD team may not just be part of a market experiencing cost pressures (competition), but may actually be the ones driving that change.
Technology	Change in technology or method used	A team may actively change hardware, software or methods to add more value for the client, instead of just being a passive victim of such change.
Social Factors	Changing workforce expectations	By empowering the workforce, and allowing them to shape and mould what the team does, their changing expectations may directly impact the project.
Overhead	Imposed changes	Instead of just adhering to changes that come down from above, a system should be in place where the ISD team can actually drive change back out through the whole enterprise.

Fuente: Conboy y Fitzgerald, 2004

- **Pro-Acción (Antes del cambio)**

Pro-acción reconoce que incluso si el cambio no se puede crear, los pasos se pueden tomar para predecir el cambio, reducir al mínimo sus efectos negativos y maximizar el potencial de beneficiarse de ella. La tabla muestra la pro-acción en el desarrollo

Tabla 2. Tabla que muestra ejemplos de pro-acción

Source of Change	Type of Change	ISD Example
Customer	Changes in Demand	The ISD team may frequently interview clients and users, or may conduct prototyping sessions, to ensure that the inevitable requirement changes are elicited as soon as possible.
Competition	Increased Cost Pressures from Competitors	The ISD team may conduct market research on other projects, engagements and product offerings to ensure their projects are not conducting unnecessarily costly activities.
Technology	Change in technology or method used	An ISD team may spend extra time during design and development to ensure the IS can support multiple platforms and platform versions.
Social Factors	Changing workforce expectations	Regular feedback from the members of the team will help to catch any concerns, issues and expectations that emerge as the project progresses.
Overhead	Imposed changes	The ISD team may negotiate a lead time for the proposed change to be implemented, reducing the impact of the change as it can be phased in gradually.

Fuente: Conboy y Fitzgerald, 2004

- **Reacción (En respuesta al cambio)**

En desarrollo software, se refiere a la reacción rápida, barata y eficaz que el equipo pueda responder al cambio.

Tabla 3. Tabla con ejemplos de reacción

Source of Change	Type of Change	ISD Example
Customer	Changes in Demand	The ISD team may have access to the code libraries from other projects at their disposal, in order to have some of the work done should certain requirements come in.
Competition	Increased Cost Pressures from Competitors	The ISD team may have outsourcing options in place so as to be able to compete on price if necessary.
Technology	Change in technology or method used	The ISD team may ensure there are trainers on hand to skill-up the staff on any new technology that emerges.
Social Factors	Changing workforce expectations	The team structure can be left somewhat flexible so people can be assigned to new roles if problems arise.
Overhead	Imposed changes	The ISD team may bring in senior officials from head office to make the transition to new procedures quick and easy.

Fuente: Conboy y Fitzgerald, 2004

- **Aprendizaje (Del cambio)**

El aprendizaje en el desarrollo de software se refiere a la capacidad del equipo para reflexionar sobre lo creativo, proactivo y reactivo que han sido en el pasado, para que puedan cambiar.

Tabla 4. Tabla con ejemplo de aprendizaje.

Source of Change	Type of Change	ISD Example
Customer	Changes in Demand	The ISD team could reflect on the changing requirements log to identify underlying trends which could be used to predict future changes i.e. every 6 months a requirement comes in to extend the server capacity.
Competition	Increased Cost Pressures from Competitors	The ISD team could do an analysis of how their processes and costs have changed in relation to those of competitors.
Technology	Change in technology or method used	The ISD team manager could learn from technology adoption problems in the past, and prepare for the same events in the future.
Social Factors	Changing workforce expectations	The ISD team manger could look at reasons why past team members have left, in order to anticipate changing expectations of new staff in the future.
Overhead	Imposed changes	The ISD team manager could look at how he/she could have got involved in the decisions over the imposed changes and the input he/she could have had

Fuente: Conboy y Fitzgerald, 2004.

8.4 METODOLOGÍAS ÁGILES VS METODOLOGÍAS FORMALES

Métodos formales y métodos ágiles son dos diferentes (y casi ortogonales) respuestas a la crisis del software que se caracterizan típicamente a errores y lentitud en los procesos de desarrollo software.

Considerando que las metodologías formales tienen su centro de atención en cuestiones de exactitud del software, ignoran casi que por completo las cuestiones de rapidez en el desarrollo software.

Metodologías ágiles centra su atención por el tema de la velocidad en el desarrollo software (también conocido como ‘’time to market’’), ignorando las cuestiones de validez formal y corrección formal.

9. INGENIERÍA DE REQUISITOS

Para abarcar el tema de Ingeniería de Requisitos, tomamos en cuenta el documento basado en la tesis doctoral de Amador Duran (Duran y Bernárdez, 2000).

Desde que en 1968 se utilizó por primera vez el término ingeniería de software, han sido muchos los estudios que se han realizado para solucionar el problema de la crisis del software. Los resultados han coincidido en comprobar la necesidad de otra ingeniería dentro de la ingeniería de software: ingeniería de requisitos.

En la última década, la comunidad de ingenieros de software ha venido trabajando arduamente en plantear metodologías y técnicas que lleven a que la ingeniería de requisitos a ser la etapa previa al ciclo de desarrollo de software y que haga parte del ciclo de vida.

Estudios realizados a nivel internacional, tanto en el ambiente americano como en el europeo, han demostrado la necesidad de una ingeniería de requisitos.

Las encuestas realizadas a los directores de los proyectos que participaron en varios estudios realizados indicaron que, en su opinión, los tres principales factores de éxito eran:

- Implicación de los usuarios
- Apoyo de los directivos
- Enunciado claro de los requisitos

Mientras que los tres principales factores de fracaso eran:

- Falta de información por parte de los usuarios
- Especificaciones y requisitos incompletos
- Especificaciones y requisitos cambiantes

9.1 DEFINICIÓN DE INGENIERÍA DE REQUISITOS

La ingeniería de requisitos es una disciplina inmadura (Durán, 2000). Más aún, no hay una definición universalmente aceptada por la comunidad informática. Se presentan algunas definiciones:

- Aplicación disciplinada de principios científicos y técnicas para desarrollar, comunicar y gestionar requisitos.
- El proceso sistemático de desarrollar requisitos mediante un proceso iterativo y cooperativo de analizar el problema, documentar las observaciones resultantes en varios formatos.
- Todas las actividades relacionadas con: Identificación y documentación de las necesidades de clientes y usuarios. Creación de un documento que describe la conducta externa y las restricciones asociadas (de un sistema) que satisfará dichas necesidades. Análisis y validación del documento de requisitos para asegurar consistencia, compleción y viabilidad.

“La ingeniería de requisitos es la rama de la ingeniería de software que tiene que ver con las metas del mundo real para, funciones de, y restricciones en los sistemas software. También concierne con las relaciones de esos factores para precisar especificaciones del comportamiento del software, y su evolución sobre el tiempo y a través de familias de software” (Zave, 1997).

Se ha argumentado que la ingeniería de requisitos es un término mal empleado. Las definiciones de los libros de texto típicos de ingeniería refieren a soluciones efectivas en costo para problemas prácticos aplicando conocimiento científico (Shaw, 1990).

El reconocimiento que ha tenido la ingeniería de requisitos de sistemas software se ha debido ampliamente a la naturaleza abstracta e invisible del software, y al amplio rango y variedad de problemas que admiten soluciones de software.

Las herramientas y técnicas utilizadas en la ingeniería de requisitos caen sobre una gran variedad de disciplinas, y se espera que la ingeniería de requisitos controlen las habilidades de diferentes disciplinas.

Como puede inferirse, no hay una unanimidad con respecto a una definición que sea precisa sobre ingeniería de requisitos.

Es claro entonces, que toma mayor importancia en concentrarse en una ingeniería de requisitos con el fin de asegurar la calidad del producto final.

9.2 EL CONCEPTO DE REQUISITO

Muchos académicos han utilizado requerimiento y requisito como sinónimos, y se presenta controversia con respecto a su uso. Para dilucidar esto, se presentan las definiciones que trae el Diccionario de la Real Academia de la Lengua:

REQUERIMIENTO- m. Acción y efecto de requerir. Der. Acto judicial por el que se intima que se haga o se deje de ejecutar una acción. Der. Aviso, manifestación o pregunta que se hace, generalmente bajo fe notarial, a alguna persona exigiendo o interesando de ella que exprese y declare su actitud o su respuesta.

Por otro lado, define: REQUISITO- m. Circunstancia o condición necesaria para una cosa

Al tener estos dos significados, es claro que el término que mejor se acuña es el de requisito. Por lo tanto, estaremos siempre hablando de requisito y no de requerimiento.

La IEEE (1990) define requisito como:

- Una condición o capacidad que un usuario necesita para solucionar un problema o lograr un objetivo.
- Una condición o capacidad que debe tener un sistema o un componente de un sistema para satisfacer un contrato, una norma, una especificación u otro documento formal.
- Una representación en forma de documento e una condición o capacidad expresadas en los dos ítem anteriores.

9.3 PROPIEDADES DESEABLES DE LOS REQUISITOS

Para que se de una buena especificación de requisitos, esta debe ser:

- Comprensible por clientes y usuarios.

Durán enfatiza que la mejor forma de comunicación es pensar en la audiencia a la que van dirigidos los requisitos, por lo que propone hacer referencia a requisitos-C, para clientes y usuarios y requisitos-D para desarrolladores.

- Correcta.

La condición necesaria para que la especificación sea correcta, es que todo requisito que esté contenido en ella representa alguna propiedad requerida por el sistema a desarrollar.

- No ambigua

Se dice de una especificación donde todo requisito tiene una sola interpretación.

- Completa

En IEEE 1996, se considera que una especificación es completa si incluye todos los requisitos identificados por el cliente y todos los requisitos necesarios para la definición del sistema.

- Consistente

La consistencia hay que verla bajo dos puntos de vista: el externo, si todo requisito contenido en la especificación no está en conflicto con otros documentos de nivel superior; el interno, si y sólo si no existen entre los requisitos que contiene.

- Modificable

Una especificación de requisitos es modificable si y sólo si su estructura y estilo de redacción permite que los cambios se puedan hacer en forma fácil, completa y consistente.

9.4 REQUISITOS NO FUNCIONALES

Algunos tipos de requisitos no funcionales más usados son los siguientes:

- Requisitos de comunicaciones del sistema

Son requisitos de carácter técnico relativos a las comunicaciones que deberá soportar el sistema software a desarrollar.

- Requisitos de interfaz de usuario

Este tipo de requisitos especifica las características que deberá tener el sistema en su comunicación con el usuario.
Se debe ser cuidadoso con este tipo de requisitos, ya que en esta fase de desarrollo todavía no se conocen bien las dificultades que pueden surgir a la hora de diseñar e implementar las interfaces, por esto no es conveniente entrar en detalles demasiado específicos.

- Requisitos de fiabilidad

Los requisitos de fiabilidad deben establecer los factores que se requieren para la fiabilidad del software en tiempo de explotación. La fiabilidad mide la probabilidad del sistema de producir una respuesta satisfactoria a las demandas del usuario.

- Requisitos de entorno de desarrollo

Este tipo de requisitos especifican si el sistema debe desarrollarse con un producto específico.

- Requisitos de portabilidad

Los requisitos de portabilidad definen qué características deberá tener el software para que sea fácil utilizarlo en otra máquina o bajo otro sistema operativo.

10. METODOLOGÍA DE ANÁLISIS PARA EL TEAM PROCESS SOFTWARE (TSP)

Para el tema del TSP, nos basamos en el libro ‘’A gap analysis methodology for the TSP’’ (Amaral, 2010).

Durante los años, la calidad en el software se está convirtiendo cada vez más en un elemento importante en la ingeniería de software.

TSP fue creado por el instituto de ingeniería de software (SEI) con el objetivo de ayudar a los ingenieros de software a mejorar la gestión de los procesos en la organización.

Este documento investigado, presenta una metodología para la evaluación de una organización versus prácticas de TSP para que sea posible evaluar los logros y las necesidades futuras que una organización tendrá durante y después de la implementación de TSP.

Este análisis tiene dos pilares en términos de colección de datos: entrevista y análisis de documentación.

Los cuestionarios se han desarrollado para guiar al equipo de evaluación en la tarea de conducir las entrevistas y más orientado en qué y en donde buscar la información en una organización.

Un modelo para la clasificación también ha sido desarrollado basado en el conocimiento y la experiencia de haber trabajado en muchas organizaciones de calidad en el software.

Una plantilla de informe también fue creada para documentar las conclusiones. El objetivo de esta metodología es ser rápido y barato en comparación con esos modelos y estándares de SEI y TSP.

10.1 ESTADO DEL ARTE

En esta sección se describe la metodología TSP que es un punto clave en este trabajo. Este documento investigativo también describe las dos metodologías más comunes y usadas en la evaluación de procesos software en una organización. Se basan en la mayoría de los modelos para la mejora de procesos de software: CMMI e ISO 15504.

10.2 TSP

El éxito de las organizaciones que producen aplicaciones de software, dependen de cuan bien están implementado los procesos de desarrollo. La implementación de métodos disciplinados de software es a menudo un desafío.

TSP junto al PSP fueron diseñados para proporcionar una estrategia y un conjunto de procedimientos operacionales para el uso de métodos disciplinados de software, tanto a nivel individual como grupal.

Países como México por ejemplo, están apostando a ser más eficientes en el desarrollo de software mediante la introducción de una iniciativa nacional de TSP.

El TSP está diseñado para guiar al equipo de desarrollo mientras ellos diseñan y desarrollan sistemas software intensivos.

TSP guía a los equipos de desarrollo y su gestión en la planificación y el desarrollo de productos de calidad en horarios predecibles. Proporciona una guía detallada y

los scripts de los procesos para que desarrolladores pongan en marcha sus operaciones de los equipos.

TSP ha demostrado ser un método eficaz para lograr productos de buena calidad con una mejora impresionante en términos de las variables claves de las organizaciones de desarrollo.

11. CMMI Y SCAMPI

CMMI es un modelo de mejora de procesos para el desarrollo de productos y servicios. Se compone de las mejores prácticas de desarrollo y mantenimiento que cubre el ciclo de vida del producto.

El propósito de CMMI es de ayudar a las organizaciones a mejorar su forma de desarrollo, mantenimiento y compra de software. El modelo CMMI es adoptado en todo el mundo y cuenta con tres constelaciones, adquisiciones y servicios de desarrollo.

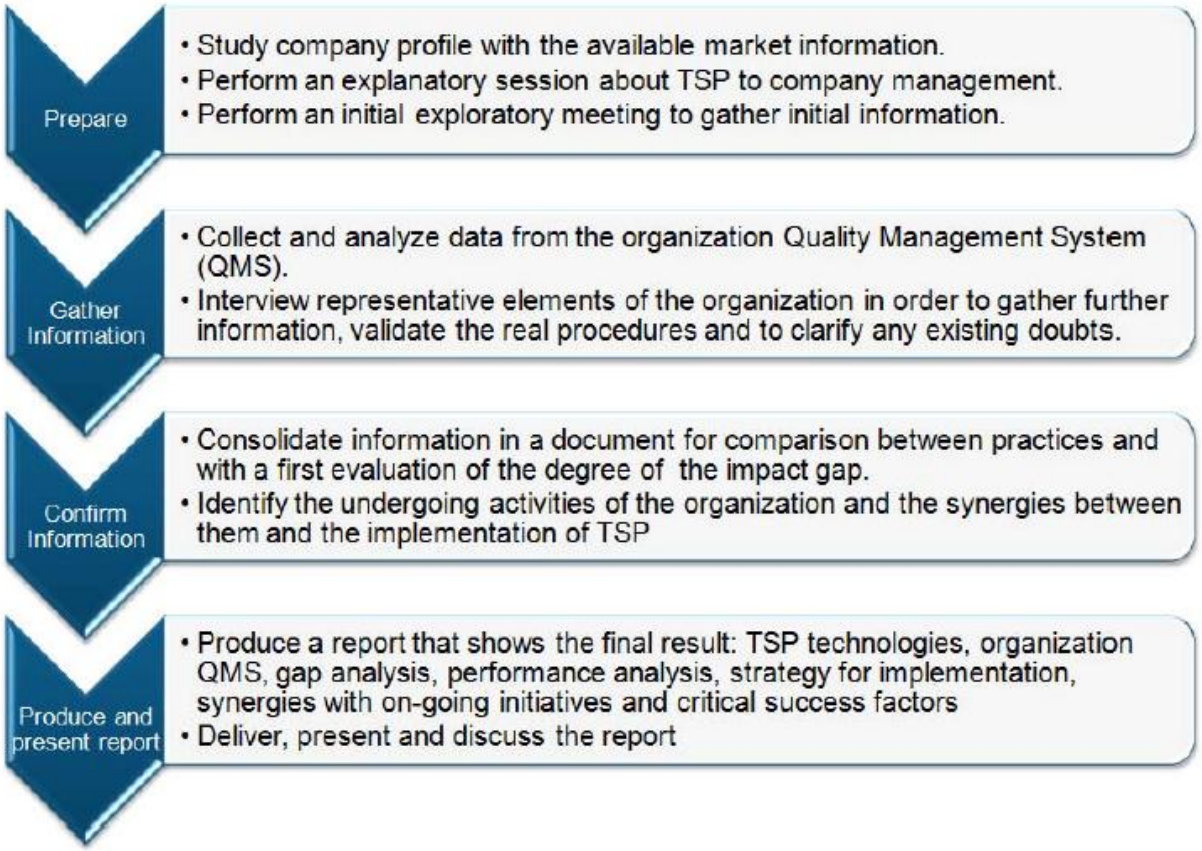
El principal objetivo del método SCAMPI es identificar los puntos débiles y fuertes de la organización, así como las calificaciones en relación con el modelo de referencia CMMI. Incluye las mejores prácticas encontradas en la comunidad después de muchos años de perfeccionamiento del modelo y del método de evaluación.

11.1 METODOLOGÍA PROPUESTA

Tanto SCAMPI e ISO 15504 no son directamente aplicables para la realización de un análisis de TSP, pero proporcionan un conjunto de elementos que se utilizaron como base para el desarrollo de una metodología de evaluación específico para las prácticas de TSP. El método desarrollado para llevar a cabo el análisis recoge datos de tres instrumentos diferentes. Utiliza la colección de artefactos, entrevistas con personas de la organización, así como conversaciones formales.

La figura 5 muestra las principales fases y actividades propuestas para el trabajo de este análisis.

Figura 5. Fases y actividades propuesta por CMMI



Fuente: Wikipedia, 2012

La ejecución exitosa de este análisis depende de tener un equipo bien preparado y entrenado, además de tener el apoyo de la organización en la disposición de los recursos (personas e información) para el equipo. Estas cosas no ocurren por accidente, se requiere una buena planificación y preparación, así como el apoyo.

Uno de los elementos clave de este proceso es evitar el uso de grandes cantidades de datos mediante la simplificación el uso de los datos disponibles y el uso de las entrevistas para validar los datos recogidos.

12. ROLES EN EL DESARROLLO SOFTWARE

Para la información con respecto a los roles que posee un proyecto de desarrollo software, nos hemos basado en el documento ‘’Formación de roles y buenas prácticas en el trabajo por la calidad del ingeniero’’ (López et al, 2011).

El desarrollo de software es una actividad que, dada su complejidad, debe desarrollarse en grupo. Además, esta actividad requiere de distintas capacidades, las que no se encuentran todas en una sola persona. Por ello, se hace necesario formar el grupo de desarrollo con las personas que cubran todas las capacidades requeridas.

Cada una de esas personas aportará al grupo parte del total de las capacidades necesarias para llevar a cabo con éxito el desarrollo. Por ello, es que cada persona debe tener un rol dentro del grupo, que viene dado por su experiencia y capacidades personales.

Estos roles son administrador de proyecto, analista, diseñador, programador, tester, asegurador de calidad, documentador, ingeniero de manutención, ingeniero de validación y verificación, administrador de la configuración y por último, el cliente. 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.

Hay que señalar que es posible que no se requieren todos los roles en un desarrollo. Eso dependerá del tamaño y del tipo del desarrollo. Por ejemplo, el desarrollo de un sistema de información de gran tamaño requerirá más roles que uno de menor tamaño. 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. No obstante, es imprescindible que dichas personas conozcan completamente todas sus tareas.

12.1 ADMINISTRADOR DE PROYECTO

El administrador de proyecto es la persona que administra y controla los recursos asignados a un proyecto, con el propósito de que se cumplan correctamente los planes definidos. Los recursos asignados pueden ser recursos humanos, económicos, tecnológicos, espacio físico, etc. En un proyecto, siempre debe existir un administrador. No obstante, un administrador puede dirigir más de un proyecto.

El administrador no es dueño de nada, es sólo un administrador temporal de los recursos. Como no es dueño de nada, debe dejarlos en la misma o mejor condición de cómo los recibió. Por ello, el foco de una buena administración debe estar en el control y coordinación de los diferentes eventos y actividades de un proyecto. Adicionalmente, deben crearse las mejores condiciones posibles para que se realicen las actividades.

12.1.1 Objetivos

Algunos de los objetivos de un administrador de proyecto son los siguientes:

- Tener el producto “a tiempo”, “bajo presupuesto” y con los requisitos de calidad definidos.
- Terminar el proyecto con los recursos asignados.
- Coordinar los esfuerzos generales del proyecto, ayudando a cada uno de sus integrantes a cumplir sus objetivos particulares. Al final, se cumplirá el objetivo general.

- Cumplir con éxito las diferentes fases de un proyecto, utilizando herramientas de administración.
- Cumplir con las expectativas del cliente.

12.1.2 Relación con otros roles

El administrador de proyecto debe relacionarse con todo el equipo de trabajo. Para ello, debe darle apoyo con lo siguiente:

- Una carta de organización de todo el proyecto.
- Un plan de trabajo general.
- Estimaciones de horas-hombre de cada actividad.

El administrador deberá tener una comunicación fluida con cada miembro del equipo para analizar problemas particulares, y si es necesario, tomar acciones correctivas.

12.2 ANALISTA

La palabra “análisis” se refiere a una característica típicamente relacionada con la inteligencia humana. Esta se refiere a la habilidad de poder estudiar un problema de una complejidad determinada, descomponiendo el problema en subproblemas de menor complejidad. De esa forma, la solución del problema completo se obtiene como la suma de las soluciones de los subproblemas de menor complejidad.

Como el experto en el problema es el cliente, se hace necesario trabajar junto a él para realizar la especificación correctamente. Los miembros del grupo que

trabajan con el cliente para realizar el análisis y especificación del sistema a construir son precisamente los analistas.

Para que el trabajo de los analistas tenga sentido para todos los integrantes del grupo, se hace necesario ponerse de acuerdo en la forma como se realizará la especificación, así como la forma como el resto del grupo la entenderá.

Una de las razones más frecuentes del fracaso de un desarrollo de software es la de realizar un análisis pobre. Debido al insuficiente esfuerzo dedicado a conocer y especificar el sistema que desea el cliente, los desarrolladores construyen sistemas que no cuentan con las características que el cliente desea.

12.2.1 Plan de trabajo

A continuación se mencionan algunas de las actividades a realizar por los analistas, las que forman parte de su plan de trabajo.

- Preparar un documento con preguntas a realizar al cliente durante las entrevistas.
- Determinar las fechas de reunión con el cliente.
- Generar un documento de especificación de requisitos de usuario en base a los acuerdos alcanzados en la primera reunión.
- Estudiar la metodología de diseño.
- Explorar las herramientas CASE a utilizar.
- Generar los diagramas de arquitectura.
- Revisar los diagramas de arquitectura con los diseñadores.

- Construir el documento de requisitos de software.
- Revisar el documento con los ingenieros de aseguramiento de la calidad y cliente, realizando modificaciones de ser necesario.

12.3 DISEÑADORES

Es el encargado de generar el diseño del sistema. Entre sus funciones está:

- Generar el diseño arquitectónico y diseño detallado del sistema, basándose en los requisitos.
- Generar prototipos rápidos del sistema (con analistas y programadores) para chequear los requisitos.
- Generar el documento de diseño arquitectónico

En cada disciplina de la ingeniería, el diseño acompaña el enfoque disciplinado que se utiliza para inventar la solución de un problema, entregando así un camino entre los requisitos y la implementación.

12.3.1 Objetivos

El propósito del diseño es el de crear una estructura interna limpia y relativamente simple, también llamada a veces una arquitectura. Un diseño es el producto final del proceso de diseño. Así, una de las metas en el diseño de software es derivar una arquitectura del sistema. Esta arquitectura sirve como un marco desde el cual se conducen más actividades de diseño detallado.

Las buenas arquitecturas de software tienden a tener algunos atributos comunes. Entre ellos se pueden mencionar los siguientes:

- Se encuentran construidos en niveles de abstracción bien definidos, provistos a través de una interfaz bien definida y controlada, y construida sobre facilidades igualmente bien definidas y controladas en niveles de abstracción inferiores.
- Existe una separación clara de preocupaciones entre la interfaz y la implementación de cada nivel, haciendo posible cambiar la implementación de un nivel sin violar las suposiciones que hicieron los clientes.

12.3.2 Plan de trabajo

El plan de trabajo de los diseñadores incluye las siguientes actividades para el diseño del sistema.

- Organizar el sistema en subsistemas.
- Identificar concurrencias inherentes al problema.
- Asignar los subsistemas a procesos y tareas.
- Seleccionar un administrador de datos.
- Identificar recursos globales y determinar mecanismos de acceso.
- Elegir un enfoque para implementar el control de la ejecución.
- Considerar condiciones de borde.

12.4 PROGRAMADORES

Los programadores deben convertir la especificación del sistema en código fuente ejecutable utilizando uno o más lenguajes de programación, así como herramientas de software de apoyo a la programación. El éxito del desarrollo de software depende grandemente de conocimiento. Este conocimiento no sólo corresponde a habilidades de programación y de administración de proyectos, sino a una percepción y entendimiento de los últimos desarrollos de la industria del software.

12.4.1 Objetivos

Uno de los principales objetivos de los programadores durante su trabajo debe ser la de reducir la complejidad del software. Algunos de los beneficios que implican la reducción de la complejidad del programa son:

- Menor cantidad de problemas de testeo.
- Aumento de la productividad de los programadores.
- Aumento de la eficiencia en la manutención del programa.

Adicionalmente, otros objetivos importantes son:

- Reducir el tiempo de codificación, aumentando la productividad del programador.
- Disminuir el número de errores que ocurren durante el proceso de desarrollo.
- Disminuir el esfuerzo de corregir errores en secciones del código que se encuentran deficientes, remplazando secciones cuando se descubren técnicas más confiables, funcionales o eficientes.

- Disminuir los costos del ciclo de vida del software.

Para alcanzar estos objetivos, es importante escoger las herramientas de desarrollo apropiadas. De eso dependerá en parte poder alcanzar los objetivos, y por lo tanto, el éxito del proyecto.

12.4.2 Actividades y metas

El lenguaje de programación escogido afecta significativamente los costos, confiabilidad y rendimiento del sistema. Ningún lenguaje es ideal para todas las aplicaciones. La elección del lenguaje debe estar basada en la naturaleza de las aplicaciones (tiempo-real, incrustada, procesamiento batch, basado en Web, etc.) y en la importancia de algunos indicadores de calidad (rendimiento versus confiabilidad). La base de datos también debe ser confiable y proteger privacidad e información comercial de usuarios no autorizados.

Los estilos de codificación incluyen los nombres de variables, la forma de hacer comentarios en el código fuente, el diseño y escritura de rutinas y módulos, la creación de tipos de datos, la selección y control de estructuras y la organización de bloques de instrucciones. A veces, algunos de estos factores son determinados por la sintaxis y el paradigma de programación utilizados, existiendo estándares para lo anterior.

Las modificaciones hechas al código deben ser solicitadas por el administrador de configuración. Otras veces, las modificaciones son solicitadas por el ingeniero de testeo directamente al programador. En algunos casos, algunas modificaciones no requieren que se completen los formularios de cambio debido a que los cambios solicitados no son relevantes o no requieren aprobación del comité de cambios (por ejemplo, no modifican los requisitos de usuarios).

12.4.3 Metodología

Existen diferentes metodologías para realizar las actividades de programación. Sin embargo, todas ellas muestran un patrón común. Este patrón consiste en la exploración de herramientas y lenguajes, la determinación del estilo de programación, el desarrollo de herramientas utilitarias y rutinas comunes para administrar la entrada, salida y errores, la codificación y depuración del sistema, la escritura de la documentación técnica, y para todo el equipo de programadores, realizar revisiones personales periódicas y reuniones.

Un factor muy importante en la construcción de un sistema es el paradigma de programación utilizado. Uno de los paradigmas muy utilizados es el orientado a objetos, el cual tiene varias ventajas sobre otras metodologías de programación.

Así mismo, existen varias técnicas de diseño y análisis orientado a objetos. Dicha elección es vital para la elección del lenguaje de programación. Por ello, si la metodología de diseño utilizada es orientada al objeto, se sugiere utilizar un lenguaje orientado a objetos.

12.4.4 Plan de trabajo

El plan de trabajo de los programadores debe contener, al menos, las siguientes actividades:

- Explorar los diferentes ambientes de desarrollo.
- Explorar los diferentes lenguajes disponibles para el ambiente.
- Explorar las diferentes herramientas de desarrollo (compiladores, bases de datos, depuradores, etc.) disponibles para el lenguaje seleccionado.
- Explorar sistemas ya construidos de los cuales, el nuevo sistema será parte.

- Elegir el estilo de programación.
- Programar las herramientas utilitarias y rutinas comunes.
- Codificar y depurar.
- Testear.
- Realizar revisiones personales y reuniones.
- Escribir la documentación técnica.

12.5 TESTER

El desarrollo de un sistema de software requiere la realización de una serie de actividades de producción. En dichas actividades existe la posibilidad de que aparezcan errores humanos. Dichos errores pueden empezar a aparecer desde el primer momento del proceso. Por ejemplo, los requisitos del sistema pueden ser especificados en forma errónea o imperfecta. Por ello, el desarrollo de software considera una actividad que apoye el proceso de detección y eliminación de los errores y defectos del sistema en construcción. El objetivo del rol de téster es precisamente realizar dichas tareas.

12.5.1 Objetivos

El objetivo principal de la labor de téster es el de diseñar test que en forma sistemática, permita eliminar diferentes clases de errores, realizando esto con la mínima cantidad de tiempo y esfuerzo.

Los objetivos específicos en la labor de un téster son los siguientes:

- Aplicar métodos para diseñar casos de test efectivos.

- Construir buenos casos de test que tengan altas probabilidades de encontrar errores aún no descubiertos.
- Demostrar que las funciones del sistema parecen estar funcionando de acuerdo a sus especificaciones.
- Proveer una buena indicación de la confiabilidad del software y algunas indicaciones de la calidad del software.

12.5.2 Metodología

Los ténsters deben utilizar una metodología que en forma sistemática, organizada y estructurada, les permita detectar y corregir, los errores y defectos introducidos en el proceso de desarrollo del sistema. Típicamente, se utilizan dos técnicas para ello: el test de la caja blanca y el test de la caja negra.

Los test de caja blanca corresponden a un método de diseño de casos de test que utilizan la estructura de control del diseño procedural para derivar los casos de test.

Por otro lado, los test de caja negra se focalizan en los requisitos de usuario del sistema. De esa forma, este tipo de test permite que los ténsters generen conjuntos de datos de entrada que ejercitarán completamente los requisitos del sistema. Los test de caja negra no son una alternativa a los test de caja blanca. Más aún, corresponde a un enfoque complementario que posiblemente descubra una clase diferente de errores.

12.5.3 Plan de trabajo

El plan de trabajo del téster debe incluir, al menos, las siguientes actividades:

- Participar en la revisión de los requisitos del sistema.
- Construir un plan de testeo.
- Coordinarse con los diseñadores para incluir el test del diseño en el documento.
- Ejecutar los test de bajo nivel.
- Ejecutar los test de mediano nivel.
- Ejecutar los test de alto nivel.
- Construir la documentación del proceso de test.

12.6 ASEGURADORES DE LA CALIDAD

En la actualidad, los factores dominantes en la administración de proyectos de software son los tiempos y costos de desarrollo. Existen buenas razones para ello. Los tiempos y costos de desarrollo son con frecuencia, muy grandes. Por ello, la administración se ha concentrado en tratar de resolver dichos problemas. Sin embargo, existe un gran peligro en esto. En la medida que crece la presión por cumplir con las fechas estipuladas, y reducir los costos, es la calidad del producto la que sufre. Cuando se acelera el desarrollo de un sistema que está atrasado, generalmente se corta todo lo que no se considere “esencial”, usualmente cortando las actividades de verificación y testeo, resultando en un producto de calidad reducida.

Para ello, debe existir el convencimiento individual y de la gerencia de considerar la calidad como una meta final, junto con el cumplimiento de plazos y costos.

12.6.1 Metodología

De entre las actividades del Asegurador de Calidad, la más importante es la de participar en las revisiones técnicas formales (RTF). Si estas revisiones están bien conducidas, son la forma más efectiva de encontrar, revelar y corregir errores mientras aún es barato encontrarlos y arreglarlos.

Los objetivos de las RTF's son: 1. Descubrir errores en funciones, lógica e implementación en cualquiera de las representaciones del software; 2. Verificar que el software bajo revisión cumple con los requisitos; 3. Asegurarse que el software ha sido representado de acuerdo al estándar en uso; 4. Alcanzar software que es desarrollado en forma uniforme; y 5. Hacer el proyecto más manejable.

12.6.2 Plan de trabajo

El plan de trabajo de un asegurador de calidad es extenso, ya que debe estar involucrado en todas las fases del desarrollo del software.

12.7 DOCUMENTADOR

Durante el proceso de desarrollo de software, se genera una gran cantidad de documentación. Dicha documentación debe ser almacenada en el repositorio del proyecto. La documentación sirve, entre otras cosas, para conocer la historia del proyecto. Hay que destacar que los documentos no se escriben al final del proyecto, sino que se van generando junto con las diferentes fases del proyecto. A medida que el proyecto va avanzando, los documentos deben ir siendo modificados para mantener el estado de los documentos a la par con el estado de desarrollo del proyecto. Sin embargo, el objetivo principal de la documentación es de actuar como medio de comunicación entre los miembros del equipo, incluyendo el cliente.

La calidad de la documentación generada es de gran importancia, debido a que la utilidad del sistema se degrada si no hay información adecuada de cómo usar o entender sus características. Para obtener esta calidad, la documentación del proyecto debe seguir estándares.

12.7.1 Objetivos

El objetivo principal del rol de documentador es el de mantener la información generada durante el proceso de desarrollo. Como objetivos específicos se tienen los siguientes:

- Permitir el almacenamiento y recuperación de la documentación de los procesos y productos más recientes durante el desarrollo, manteniendo así la información al día.
- Asegurarse que los cambios que necesitan hacerse en el sistema serán reflejados en la documentación correspondiente.
- Construir el manual de usuarios del sistema, MUS, que contempla los aspectos de uso del sistema.

12.7.2 Metodología

La metodología de trabajo del documentador está enfocada a realizar las acciones que le permita cumplir con sus objetivos principales y específicos. Al inicio, debe establecer los formatos usados en los documentos del proyecto, definiendo las estructuras de los documentos:

- Información de identificación de cada documento (nombre, tipo, autores, versión, etc.).

- Información que facilite la recuperación de la información (resúmenes, palabras claves).

- Enfoques para estructuras capítulos, secciones y subsecciones, y su numeración respectiva.

- Índices y numeración de páginas.

Hay que señalar que no es poco frecuente encontrar gente que piensa que lo importante de un documento es su contenido, menospreciando su forma. Es correcto pensar que el contenido es vital. Sin embargo, se olvidan que un documento es un instrumento de comunicación, y como tal, debe trabajarse su forma para no tener ambigüedades, inconsistencias, y malas interpretaciones. Lo anterior sugiere la necesidad de trabajar las formas del lenguaje con el propósito de producir documentos con mayor capacidad comunicacional.

12.7.3 Herramientas de apoyo

El documentador requerirá herramientas para la elaboración de documentos (minutas, documentos de acuerdos, manual de usuario) y herramientas de apoyo para la elaboración y administración del repositorio de documentos. Dentro del primer conjunto de herramientas se encuentran las siguientes:

- Editor de texto, que permite elaborar documentos fácilmente y almacenarlos en diferentes formatos, incluyendo HTML. Debe incluir una herramienta para el chequeo ortográfico y de gramática.

- Navegador Web y editor HTML, que permita editar y publicar documentos HTML, proveyendo diversas funciones.

12.7.4 Plan de trabajo

Las actividades del documentador se dividen en dos grupos. El primer grupo se refiere a actividades que se realizan durante el ciclo de vida del proyecto. Entre estas actividades se encuentra la elaboración de las actas de reuniones, el almacenamiento de los documentos generados durante el proyecto, manutención del repositorio de información, y manutención del sitio Web del proyecto.

El segundo grupo se refiere a actividades que se realizan sólo una vez, al principio o al final del proyecto. Entre estas actividades se encuentra la elaboración de la documentación, el diseño e implementación de la base de datos para la documentación, y la elaboración de los perfiles de acceso a la documentación para cada uno de los miembros del equipo.

13. TESTING DE SOFTWARE

Para el tema del Testing en el desarrollo software, hemos hecho el análisis según la investigación hecha por Maximiliano Cristia (Cristia 2009).

‘’El testing de programas puede ser una forma muy efectiva de mostrar la presencia de errores, pero es desesperanzadoramente inadecuado para mostrar su ausencia’’. Edsger Dijkstra.

‘’Esté atento a los errores en el código mostrado más arriba; yo solo demostré que es correcto, pero no lo ejecute’’. Donald Knuth.

13.1 VERIFICACION Y VALIDACION

El testing de software pertenece a una actividad o etapa del proceso de producción de software denominada Verificación y Validación.

V&V es el nombre genérico dado a las actividades de comprobación que aseguran que el software respeta su especificación y satisface las necesidades de sus usuarios. El sistema debe ser verificado y validado en cada etapa del proceso de desarrollo utilizando los documentos (descripciones) producidos durante las etapas anteriores.

En rigor no sólo el código debe ser sometido a actividades de V&V sino también todos los subproductos generados durante el desarrollo del software.

Si bien estos términos en su uso cotidiano pueden llegar a ser sinónimos, en Ingeniería de Software tienen significados diferentes y cada uno tiene una definición más o menos precisa.

- Validación: > ¿estamos construyendo el producto correcto?
- Verificación: > ¿estamos construyendo el producto correctamente?

En este sentido, la verificación consiste en corroborar que el programa respeta su especificación, mientras que validación significa corroborar que el programa satisface las expectativas del usuario

13.2 DEFINICION DE TESTING

El testing es una actividad desarrollada para evaluar la calidad del producto, y para mejorarlo al identificar defectos y problemas. El testing de software consiste en la verificación dinámica del comportamiento de un programa sobre un conjunto de casos de prueba, apropiadamente seleccionados a partir del dominio de ejecución que usualmente es infinito, en relación con el comportamiento esperado.

Es una técnica dinámica en el sentido de que el programa se verifica poniéndolo en ejecución de la forma más parecida posible a como ejecutará cuando esté en producción - esto se contrapone a las técnicas estáticas las cuales se basan en analizar el código fuente.

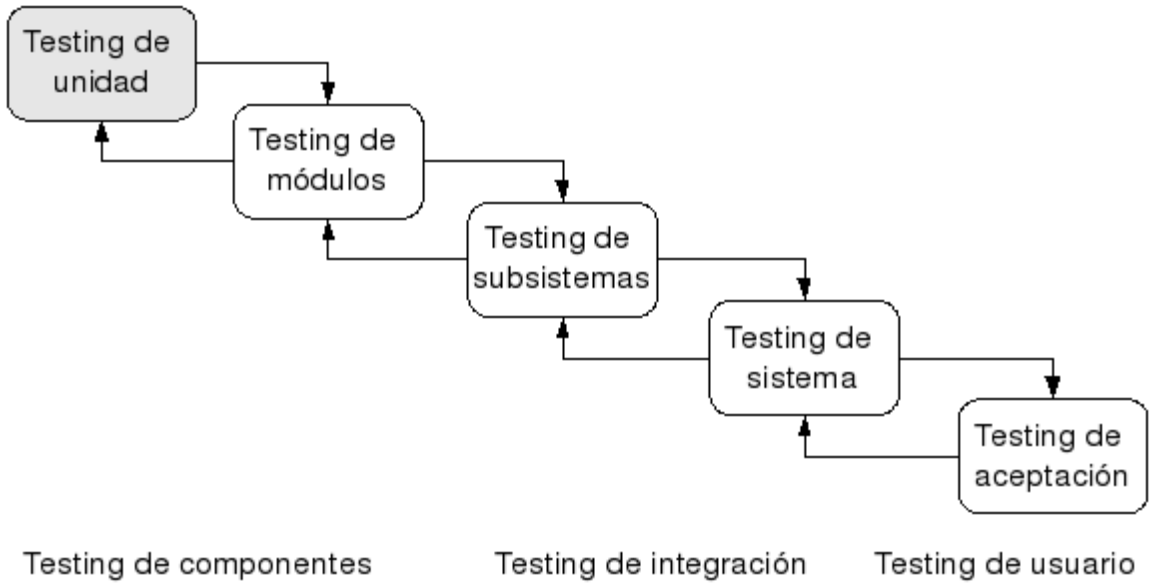
Aunque en la industria de software, el testing es la técnica predominante, en el ambiente académico se estudian muchas otras técnicas.

De acuerdo a la definición clásica de corrección, un programa es correcto si verifica su especificación. Entonces, al considerarse como técnica de verificación el testing, un programa es correcto si ninguno de los casos de prueba seleccionados detecta un error.

13.3 EL PROCESO DE TESTING

Es casi imposible, excepto para los programas más pequeños, testear un software como si fuera una única entidad monolítica. Los grandes sistemas de software están compuestos de subsistemas, módulos y subrutinas. Se sugiere, por lo tanto, que el proceso de testing esté guiado por dicha estructura, como muestra la Figura 6.

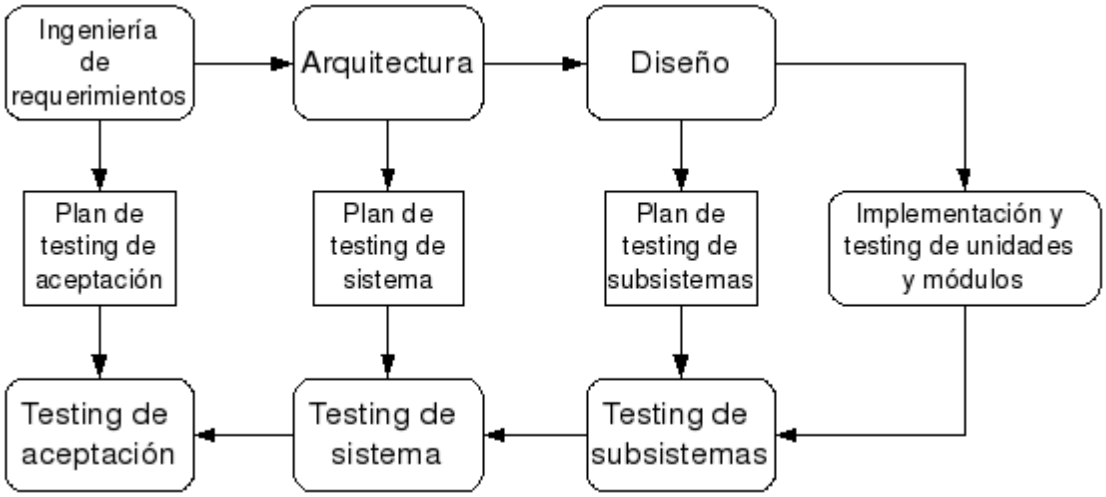
Figura 6. El proceso general de testing.



Fuente: Cristiá, 2009

Más aún, si el proceso sugerido se combina adecuadamente con el proceso de desarrollo, como muestra la Figura 7, entonces se habilita la posibilidad de ir detectando errores de implementación lo más tempranamente posible - lo que a su vez reduce el costo de desarrollo.

Figura 7. Coordinación entre el proceso de testing y el proceso de desarrollo.

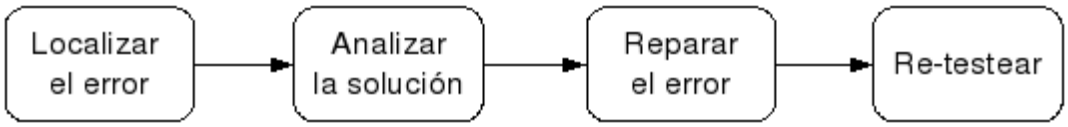


Fuente: Cristiá, 2009

Como sugiere la Figura 7, un sistema complejo suele testearse en varias etapas que por lo general se ejecutan siguiendo una estrategia bottom-up, aunque el proceso general es iterativo. Se debe volver a las fases anteriores cada vez que se encuentra un error en la fase que está siendo ejecutada. Por ejemplo, si se encuentra un error durante el testing de un subsistema particular, una vez que aquel haya sido reparado se debe testear la unidad donde estaba el error y luego el módulos al cual pertenece la unidad reparada.

En general, una vez detectado un error se sigue el proceso graficado en la Figura 8. De aquí a que las iteraciones del proceso de la Figura 7 se las llame re-testing.

Figura 8. Proceso de re-testing



Fuente: Cristiá, 2009

14. ELICITACIÓN DE REQUISITOS DE SOFTWARE

Para dar nuestras recomendaciones de mejora para la compañía, comenzaremos con el proceso de elicitación de requisitos. Para este caso, el análisis se basó en el documento ‘’Metodología para la elicitación de requisitos de sistemas software’’ (Duran y Bernárdez, 2000).

14.1 TAREAS RECOMENDADAS

TAREA 1: Obtener información, sobre el dominio del problema y el sistema actual

Los objetivos principales de esta tarea consisten en conocer el dominio del problema y la situación actual.

Es de gran importancia identificar qué es lo que hace la empresa contratante del servicio y el cómo se están haciendo las cosas, incluso mucho antes de iniciar a hacer entrevistas y modelos del sistema, a su vez de de gran valor para la organización tener claro el dominio del problema a resolver ya que este es el punto de partida para iniciar las negociaciones con el cliente.

Enfrentarse a un desarrollo sin tener conocimiento del que hace la organización contratante, y sin tener conocimiento en el dominio del problema, puede causar que el desarrollo que se entregue no cumpla con las expectativas planteadas.

Productos internos que el equipo de desarrollo necesitará para resolver esta tarea:

- Información recopilada
- Libros, artículos, folletos comerciales
- Desarrollos previos sobre el mismo dominio.

Productos entregables:

- Introducción
- Participantes del proyecto
- Descripción del sistema actual

Técnicas recomendadas a utilizar:

- Obtener información de fuentes externas al negocio del cliente
- Informes sobre el sector, consultas con expertos y demás
- En el caso de que se trate de un dominio muy específico, puede ser necesario recurrir a fuentes internas al propio negocio del cliente, en cuyo caso pueden utilizarse las técnicas auxiliares como:
    - El estudio de documentación
    - Observación in situ

TAREA 2: Preparar y realizar las sesiones de elicitación / negociación

Los objetivos de esta tarea son:

- Identificar a los usuarios participantes.
- Conocer las necesidades de los clientes
- Conocer las necesidades de los usuarios
- Resolver posibles conflictos

Teniendo en cuenta la información recopilada de la tarea 1, en esta tarea se debe preparar y realizar las reuniones con los clientes y usuarios, con el objetivo de obtener sus necesidades y resolver posibles conflictos que se hayan encontrado en iteraciones previas del proceso.

Esta tarea es especialmente crítica, y ha de realizarse con especial cuidado, ya que los clientes y posibles usuarios no saben qué necesita saber el equipo de desarrollo para llevar a cabo su labor.

De esto surge:

- Notas tomadas durante las reuniones.
- Transcripciones o actas de reuniones.
- Grabaciones en cinta o video de las reuniones.
- Formularios y demás.

Productos entregables:

- Participantes de los proyectos.
- Usuarios participantes.
- Requisitos o conflictos que fueron identificados claramente en las sesiones de elicitación.
- Objetivos.

Técnicas recomendadas:

- Técnicas de elicitación de requisitos.
- Plantillas de objetivos.
- Técnicas de negociación Win-Win.

TAREA 3: Identificar o revisar los objetivos del sistema.

Identificar los objetivos que se esperan alcanzar, revisar en el caso que se encuentren conflictos los objetivos previamente identificados.

A partir de la información obtenida en la tarea anterior, en esta tarea se deben identificar qué objetivos se esperan alcanzar, una vez que el sistema a desarrollar se encuentre en explotación o revisarlos en función de los conflictos previamente identificados.

Entregables:

- Objetivos del sistema.

Técnicas recomendadas:

- Análisis de factores críticos de éxito.
- Plantilla para especificar los objetivos del sistema

TAREA 4: Identificar o revisar los requisitos de almacenamiento de información

Identificar los requisitos de almacenamiento de información que deberá cumplir el sistema software a desarrollar.

Revisar, en el caso de que se encuentren conflictos, los requisitos de almacenamiento de información previamente identificados.

A partir de la información obtenida en las tareas 1 y 2, y teniendo en cuenta los objetivos y el resto de los requisitos, en esta tarea se debe identificar o revisar, si existen conflictos, que información de relevancia se debe gestionar con el cliente, y almacenar el software a desarrollar.

Inicialmente, se dividirán los conceptos generales para posteriormente ir detallándolos hasta obtener todos los datos relevantes.

Productos entregables:

- Requisitos de almacenamiento de información.

Técnicas recomendadas:

- Utilizar plantillas para documentar estos requisitos de almacenamiento de información

TAREA 5: Identificar o revisar los requisitos funcionales

Identificar los actores del sistema a desarrollar, los requisitos funcionales, casos de uso, que debería cumplir el sistema a desarrollar.

Revisar dado el caso que se encuentren conflictos, los requisitos funcionales previamente identificados.

A partir de la información obtenida en las tareas 1 y 2, y teniendo en cuenta los objetivos identificados y el resto de requisitos, en esta tarea se deben identificar si existen conflictos, que debe hacer el sistema a desarrollar con la información identificada en la tarea 4.

Inicialmente se identificaran los actores que interactúan con el sistema, es decir, aquellas personas u otros sistemas que serán los orígenes o destinos de la información que consumirá o producirá el sistema a desarrollar y que forma su entorno.

Posteriormente se deben identificar los casos de uso asociados a los actores, los pasos de cada caso de uso y posteriormente se detallaran los casos de uso con las posibles excepciones hasta definir todas las situaciones posibles.

Productos entregables:

- Requisitos funcionales

Técnicas recomendadas:

- Casos de uso
- Plantillas para actores
- Plantillas para los requisitos funcionales.

TAREA 6: Identificar o revisar los requisitos no funcionales

Identificar los requisitos no funcionales.

A partir de la información obtenida en las tareas 1 y 2, y teniendo en cuenta los objetivos identificados y el resto de los requisitos, en esta tarea se deben identificar o revisar si existen o no conflictos, los requisitos no funcionales, normalmente de carácter técnico o legal

Productos entregables:

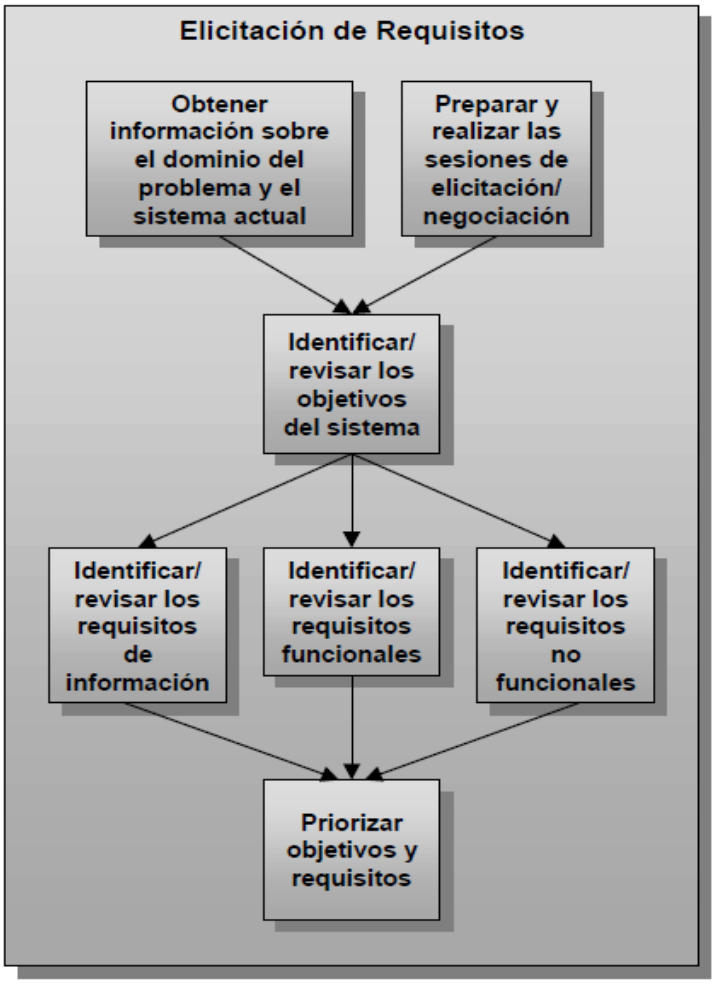
- Requisitos no funcionales

Técnicas recomendadas:

- Plantilla para requisitos no funcionales

El orden recomendado de realización para estas tareas es: 1. .7, aunque las tareas 4, 5, y 6 pueden realizarse simultáneamente o en cualquier orden que se considere oportuno (ver figura 9). La tarea 1 es opcional y depende del conocimiento previo que tenga el equipo de desarrollo sobre el dominio del problema y el sistema actual.

Figura 9. Tareas de elicitacion de requisitos



Fuente: Duran y Bernárdez, 2000

15. CONFORMACIÓN DE EQUIPOS DE TRABAJO

Para la conformación de equipos, hemos realizado el análisis según varios documentos. El primero, es el libro de Humphrey ‘’Leadership, teamwork, and trust’’ y de ‘’A gap analysis methodology for the TSP.

El trabajar en equipos interdisciplinarios, es trabajar dentro de un sistema, ya que cada persona conforma un elemento y cada persona está asociada a otra para buscar una solución a un problema específico, ya que entre personas es mucho más fácil que se genere el caos dentro del sistema, debido a las inconstantes necesidades y fricciones humanas.

El equipo de trabajo es el conjunto de personas asignadas o auto asignadas, de acuerdo a habilidades y competencias específicas, para cumplir una determinada meta bajo la conducción de un coordinador.

En los equipos de trabajo debe haber una serie de características, las cuales harán que se pueda alcanzar el objetivo común de forma armónica:

- Debe existir una integración armónica de funciones y actividades desarrolladas por diferentes personas.
- Para su implementación requiere que las responsabilidades sean compartidas por sus miembros.
- Necesita que las actividades desarrolladas se realicen en forma coordinada.
- Necesita que los programas que se planifiquen en equipo apunten a un objetivo común.

Para que un grupo se transforme en un equipo es necesario favorecer un proceso en el cual se exploren y elaboren aspectos relacionados con los siguientes conceptos:

- Cohesión
- Asignación de roles y normas.
- Comunicación.
- Definición de objetivos.
- Interdependencia.

La cohesión se refiere a la atracción que ejerce la condición de ser miembro de un grupo. Los grupos tienen cohesión en la medida en que ser miembro de ellos sea considerado algo positivo y los miembros se sienten atraídos por el grupo.

Todos los grupos asignan roles a sus integrantes y establecen normas aunque esto no se discuta explícitamente. Las normas son las reglas que gobiernan el comportamiento de los miembros del grupo. Atenerse a roles explícitamente definidos permite al grupo realizar las tareas de modo eficiente

Una buena comunicación interpersonal es vital para el desarrollo de cualquier tipo de tarea. Los grupos pueden tener estilos de funcionamiento que faciliten o que obstaculicen la comunicación. Se pueden realizar actividades en donde se analicen estos estilos. Algunos especialistas sugieren realizar ejercicios donde los integrantes deban escuchar a los demás y dar y recibir información.

Es muy importante que los integrantes del equipo tengan objetivos en común en relación con el trabajo del equipo y que cada uno pueda explicitar claramente

cuáles son sus objetivos individuales. Para ello se sugiere asignar a los grupos recién formados la tarea de definir su misión y sus objetivos, teniendo en cuenta que los objetivos compartidos son una de las propiedades definitorias del concepto "equipo".

El aprendizaje colaborativo se caracteriza por la interdependencia positiva entre las personas participantes en un equipo, quienes son responsables tanto de su propio aprendizaje como del aprendizaje del equipo en general.

Para que los integrantes tomen conciencia y experimenten lo que significa la interdependencia, algunos docentes sugieren poner en práctica un ejercicio denominado "Supervivencia en una isla" en el que los compañeros de equipo deben imaginar cuáles son los elementos que necesitan para sobrevivir en una isla desierta luego de un naufragio. Luego, deben realizar el mismo análisis de modo grupal. En general, los rankings grupales suelen ser más precisos que la mayoría de los individuales.

Un equipo necesita objetivos claros. No basta saber lo que se debe hacer en cada momento, sino para qué se hace, cuál es la misión general del equipo y saber que los objetivos compartidos producen compromisos. Un equipo debe examinar rutinariamente lo que hace. Si aparecen problemas de rendimiento/productividad o de calidad, se deben resolver antes de que sean importantes.

15.1 ETAPAS PARA CREAR UN EQUIPO

La conformación de un equipo de propiedad intelectual gira alrededor de las personas, sus relaciones y emociones. ¿Cómo mejorar la relación entre las personas? ¿Cómo entender lo que nos pasa? ¿Hacia dónde vamos? ¿Qué queremos para nosotros? ¿Y para el equipo?

Nuestro objetivo será encontrar una Visión Compartida por el equipo: aquello que una y guíe al equipo más allá de la duración de un proyecto. Algo que sea de tanto valor para el equipo y para las personas que logre perdurar en el tiempo. Algo por lo que valga la pena el esfuerzo.

Las etapas por las que pasa un equipo son:

- **Check-In**: las personas del equipo toman consciencia de su estado. Se valora la presencia intelectual. La integridad es un valor fundamental para el equipo. Se genera conexión entre las personas.

- **Decisión**: el equipo decide de forma unánime. Se logra la Responsabilidad Total. Se actúa con intención. Aparece una intención y un comportamiento alineados.

- **Alineamiento personal**: cada integrante tiene un objetivo personal, sabe lo que quiere y acciona para lograr. Se logra el apoyo del equipo para los alineamientos personales, de forma de crear una verdadera unidad de equipo.

- **Visión compartida**: el equipo genera una Visión, una meta a largo plazo que trasciende al proyecto y guía todo su comportamiento.

16. PERFILES DEL EQUIPO DE TRABAJO

Para el análisis de los roles en el desarrollo software, tomamos en cuenta el perfil que debe tomar cada personas a la que se le asigna un rol en específico. Esta investigación fue sacada del libro ‘’Apuntes de taller de ingeniería de software’’ (Fuller, 2003).

16.1 ADMINISTRADOR DEL PROYECTO (GERENCIA)

- Perfil de un administrador de proyecto

El administrador de proyecto deberá tener, al menos, las siguientes capacidades personales para desarrollar adecuadamente su trabajo:

Abstracción: Entender y comunicar aspectos no tangibles, como visión y misión del equipo de trabajo. Deberá además, poder entender y ver el proyecto completo como una unidad y sus relaciones entre sus partes.

Concretización: Utilizando los recursos e información disponibles, obtener conclusiones y tomar acciones específicas para manejar el proyecto.

Organización: Distribuir eventos y actividades de acuerdo a los recursos y tiempos disponibles para llevar el proyecto al éxito

Liderazgo: Llevar a un equipo a lograr sus objetivos.

Experiencia: Haber estado en situaciones similares en el pasado

16.2 ANALISTA

- Perfil de un analista

Un analista es una persona con capacidades de comunicación, debido a que deberá tener un contacto estrecho con el cliente. Por lo anterior, debe ser una persona sociable, expresando sus ideas en forma clara en un lenguaje común con el cliente.

También debe tener la capacidad de escuchar y entender al cliente. Se espera que los analistas tengan un alto grado de desarrollo de su inteligencia emocional.

Los analistas deben conocer y manejar perfectamente los métodos y las tecnologías de apoyo para realizar las fases de análisis.

16.3 DISEÑADOR

- Perfil de un diseñador

El perfil de un diseñador debe incluir las siguientes características:

Para sistemas de tamaño pequeño y mediano, el diseño arquitectónico es realizado por una o dos personas calificadas. Deben mostrar habilidad inusual para sintetizar soluciones construibles por sobre un gran conjunto de restricciones.

Generalmente son los más capacitados para realizar decisiones estratégicas debido a su experiencia previa en la construcción de sistemas similares.

No son necesariamente los desarrolladores con más experiencia.

Deben tener habilidades de programación adecuadas.

Deben conocer muy bien la metodología de diseño utilizada, así como sus herramientas de apoyo.

16.4 DESARROLLADOR

- Perfil de un programador

El perfil del programador requiere conocimiento en varios ambientes, pudiendo ayudarle a los analistas y diseñadores a elegir el apropiado. Debe tener experiencia en el desarrollo de aplicaciones en el ambiente seleccionado.

Debe conocer diferentes lenguajes de programación disponibles para el ambiente seleccionado, y debe tener experiencia en el lenguaje de programación seleccionado. Las herramientas utilitarias desarrolladas en proyectos previos pueden ser útiles en el proyecto actual. Es preferible que el programador tenga conocimientos en diferentes paradigmas de programación y estilos.

16.5 TESTER

- Perfil de un téster

El perfil de un téster debe considerarse las siguientes características:

- Ser un buen programador en el lenguaje seleccionado, y tener experiencia en el desarrollo del sistema.
- Conocer bien la metodología de diseño utilizada.
- Ser sistemático en las revisiones de código y resultados de los tests.
- Tener una personalidad agresiva para buscar errores en el código y documentos del proyecto

17. PROBADORES Y PROCESOS DE PRUEBA

Para todo lo que concierne al tema de testing, nos hemos referenciado en la investigación realizada por Maximiliano Cristía y en el libro ‘’Practical Mode-Based testing: A tool approach’’ de Mark Utting y Bruno Legeard.

Los probadores de software deben considerar lo siguiente al planificar y ejecutar las pruebas: el software y su función de cálculo, las entradas y cómo se pueden combinar y el entorno en el que el software eventualmente funcionará. Este difícil proceso requiere tiempo, sofisticación técnica y una adecuada planificación. Los probadores no sólo deben tener buenas habilidades de desarrollo, sino también conocimientos en lenguajes formales, teoría de grafos, lógica computacional y algoritmos. De hecho, los probadores creativos aplican muchas disciplinas, relacionadas con la informática, al problema de las pruebas, a menudo con resultados impresionantes.

Incluso el software más simple presenta obstáculos por lo que, para tener una visión más clara acerca de algunas de las dificultades inherentes a las pruebas de software, es necesario acercarse a ellas a través de la aplicación de cuatro fases:

- Modelar el entorno del software.
- Seleccionar escenarios de prueba.
- Ejecutar y evaluar los escenarios.
- Medir el progreso de las prueba.

Estas fases le ofrecen a los probadores una estructura en la que pueden agrupar los problemas relacionados y que deben resolver antes de pasar a la siguiente fase.

17.1 MODELAR EL ENTORNO DEL SOFTWARE

La tarea del probador es simular la interacción entre el software y su entorno, para lo que debe identificar y simular las interfaces que utiliza el sistema, y enumerar las entradas que pueden circular por cada una de ellas.
Éste podría ser el asunto más importante que enfrentan y, teniendo en cuenta los diversos formatos de archivo, los protocolos de comunicación y las terceras partes disponibles puede ser muy complicado. Las interfaces más comunes son:

- Las interfaces humanas
- Las interfaces de software
- Las interfaces del sistema de archivos
- Las interfaces de comunicación

Luego, los probadores deben comprender las interacciones de usuario que están fuera del control del software bajo prueba, ya que las consecuencias pueden ser graves si el software no está preparado.

17.2 SELECCIONAR ESCENARIOS DE PRUEBA

Muchos modelos de dominio y particiones de variables representan un número infinito de escenarios de prueba, cada uno de los cuales cuesta tiempo y dinero. Sólo un subconjunto de ellos se puede aplicar en cualquier programa de desarrollo de software realista, así que ¿cómo hace un probador inteligente para seleccionar ese subconjunto? ¿17 es mejor valor de entrada que 34? Los probadores, sin

embargo, prefieren una respuesta que se refiera a la cobertura de código fuente o a su dominio de entrada y se orientan por: la cobertura de las declaraciones de código. Pero si el código y la cobertura de entrada son suficientes, los productos entregados deberían tener muy pocos errores.

En cuanto al código, no son las declaraciones individuales las que interesan a los probadores, sino los caminos de ejecución: secuencias de declaraciones de código que representan un camino de ejecución del software pero, desafortunadamente, existe un número infinito de caminos.

En cuanto al dominio de entrada, no les interesan las individuales, sino las secuencias de entrada que, en su conjunto, representan escenarios a los que el software debe responder, pero también existe un número infinito de ellas.

Las pruebas se organizan desde dichos conjuntos infinitos hasta lograr, lo mejor posible, los criterios adecuados de datos de prueba; que se utilizan adecuada y económicamente para representar cualquiera de esos conjuntos. "Mejor" y "adecuadamente" son subjetivos: los probadores típicamente buscan el conjunto que garantiza encontrar la mayoría de los errores.

17.3 EJECUTAR Y EVALUAR LOS ESCENARIOS

Una vez identificado el conjunto de casos de prueba adecuado, los probadores los convierte a formatos ejecutables, a menudo código, de modo que los escenarios de prueba resultantes simulan la acción de un usuario típico.

Debido a que los escenarios de prueba se ejecutan manualmente constituye un trabajo intensivo y, por tanto, propenso a errores, por lo que los probadores deben tratar de automatizarlos tanto como sea posible. En muchos entornos es posible aplicar automáticamente las entradas a través del código que simula la acción de los usuarios, y existen herramientas que ayudan a este objetivo. Pero la

automatización completa requiere la simulación de cada fuente de entrada, y del destino de la salida de todo el entorno operacional. A menudo, los probadores incluyen código para recoger datos en el entorno simulado, como ganchos o seguros de prueba, con los que recogen información acerca de las variables internas, las propiedades del objeto, y otros. De esta forma, ayuda a identificar anomalías y a aislar errores. Estos ganchos deben ser eliminados cuando el software se entrega.

La evaluación de escenarios, la segunda parte de esta fase, es fácil de fijar pero difícil de ejecutar. La evaluación implica la comparación de las salidas reales del software, resultado de la ejecución de los escenarios de prueba, con las salidas esperadas, tal y como están documentadas en la especificación, que se supone correcta, ya que las desviaciones son errores.

17.4 MEDIR EL PROGRESO DE LAS PRUEBAS

Medir las pruebas consiste en contar cosas: el número de entradas aplicadas, el porcentaje de código cubierto, el número de veces que se ha invocado la aplicación, el número de veces que se ha terminado la aplicación con éxito, el número de errores encontrados y así sucesivamente. Los valores de estos conteos pueden dar muy pocas luces acerca de los avances de las pruebas, y muchos probadores alteran estos datos para dar respuesta a las preguntas, con lo que determinan la completitud estructural y funcional de lo que han hecho. Por ejemplo, para comprobar la completitud estructural los probadores pueden hacerse estas preguntas:

- ¿He probado para errores de programación común?
- ¿He ejercitado todo el código fuente?
- ¿He forzado a todos los datos internos a ser inicializados y utilizados?

- ¿He encontrado todos los errores sembrados?

Estas preguntas son útiles para los probadores por ejemplo, para determinar cuándo detener las pruebas o cuándo está listo un producto para su liberación, es más complejo. Se necesitan medidas cuantitativas de la cantidad de errores que queden en el software, y de la probabilidad de que cualquiera de ellos sea descubierto por el usuario. Si los probadores pudieran lograr esta medida, sabrían cuando parar las pruebas y sería posible que se acercaran cuantitativamente al problema de forma estructural y funcional.

17.5 UNA ESTRATEGIA PARA LA ELIMINACIÓN DE DEFECTOS

Se podría argumentar que muchos de los defectos en los programas grandes están en el sistema y no en los módulos. Esto, sin embargo, no es cierto. Con pocas excepciones, los defectos en los grandes sistemas están en los módulos que lo constituyen.

Estos defectos se pueden dividir en dos clases: aquellos que implican solamente un módulo y los que implican interacciones entre módulos.

Si se sigue a cabalidad el modelo de PSP y se obtiene un alto rendimiento, se eliminará casi toda la primera clase de defectos. La segunda clase, sin embargo, es mucho más difícil. La razón es que, los sistemas grandes y complejos, implican muchas interacciones que son difíciles de visualizar para los diseñadores. Una buena forma de abordar este problema es seguir una estrategia como la siguiente:

- Realizar inspecciones de todas las interfaces de módulos y sus interacciones.
- Inspeccionar los requisitos para asegurar que todas las funciones importantes son adecuadamente entendidas, diseñadas e implementadas.

- Inspeccionar el sistema y el diseño del programa frente a los requisitos, para asegurar que son tratados adecuadamente todos los requisitos clave.
- Hacer pruebas de unidad exhaustivas después de que se haya inspeccionado el código.
- Hacer una prueba de integración global.
- Hacer pruebas a todo el sistema.

Con módulos que inicialmente tienen una alta calidad, sin embargo, se puede seguir la estrategia anteriormente mencionada. Entonces tendremos una oportunidad razonable de hacer sistemas de alta calidad. Mejor que derrochar el tiempo intentando encontrar y arreglar los defectos de los módulos, la prueba de integración se puede enfocar en probar las interfaces entre los módulos. La prueba del sistema ahora podría tratar cuestiones críticas del sistema como el rendimiento, la funcionalidad, recuperación y seguridad. Hasta que los ingenieros no logren de una forma consistente rendimientos personales altos, dichas pruebas no se harán.

18. EL PROCESO DE DESARROLLO DEL SOFTWARE

Para este capítulo, comenzamos citando a Watt S. Humphrey: “Cuando he colaborado con organizaciones de software para mejorar su rendimiento, uno de los principales problemas que he encontrado, es la determinación del rendimiento real de las mismas. Un grupo, por ejemplo, no podía decirme cuántos proyectos se habían entregado tarde o habían superado el presupuesto. Sin dichos datos, no había forma de decirles si ellos iban mejorando o empeorando” (Watts, 1995).

‘’El PSP es un marco de trabajo que ayuda a los ingenieros del software a medir y mejorar su forma de trabajar. Este modelo, ayuda a desarrollar programas y nos muestra cómo, utilizando los procesos, podemos mejorar nuestra forma de trabajar’’ (Watts, 1995).

Las fases del proceso del PSP se describen a continuación y están resumidas en la siguiente Tabla:

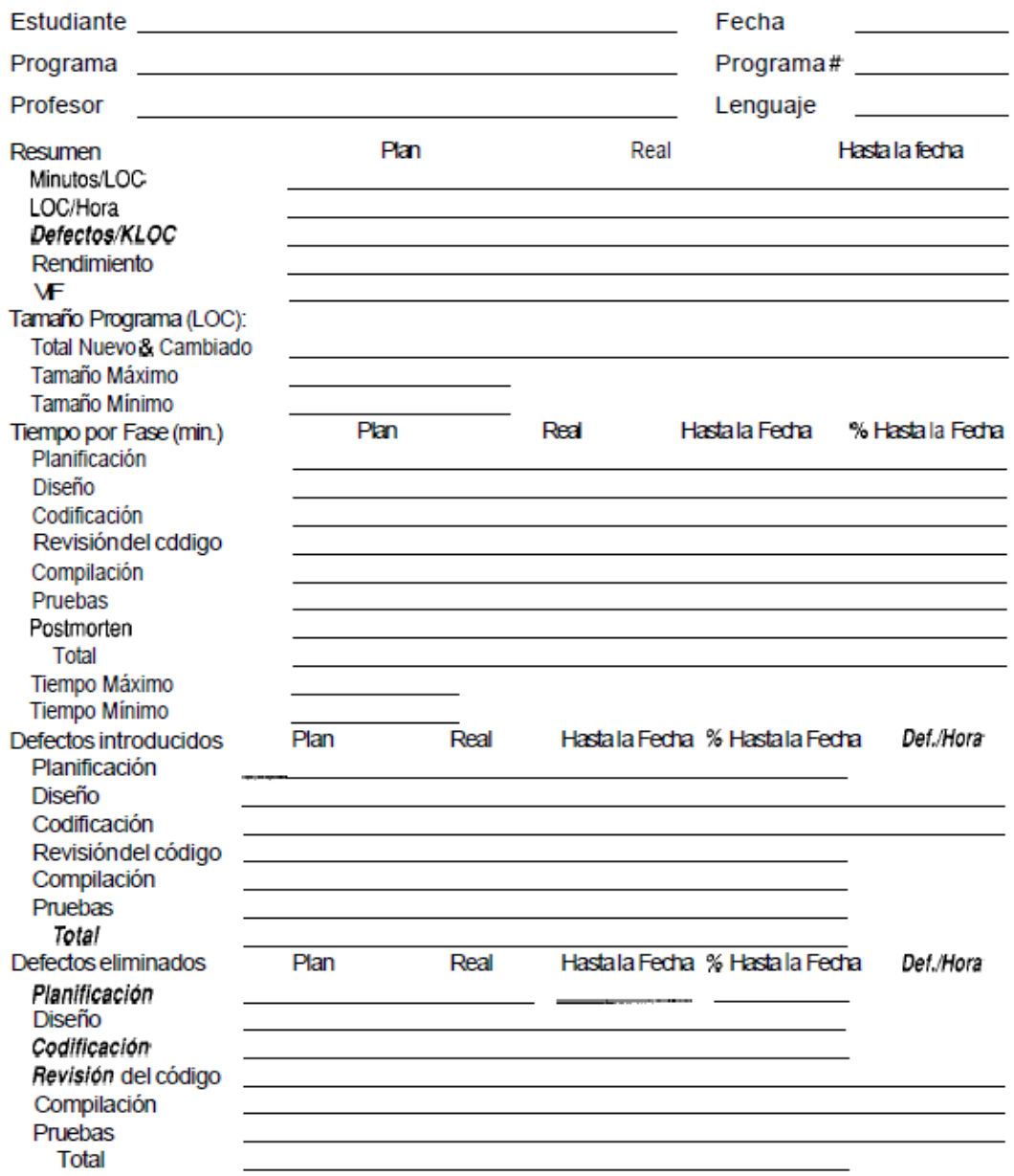
Tabla 5. Guion del proceso PSP

	Propósito	Guiar en el desarrollo de pequeños programas.
	Criterios de entrada	La descripción del problema. Tabla Resumen del Plan del Proyecto del PSP. Datos de tamaños y tiempos reales de programas anteriores. Cuaderno de Registro de Tiempos.
1	Planificación	Obtén una descripción de las funciones del programa. Estima las LOC Máx., Mín. y total requeridas. Determina los Minutos/LOC. Calcula los tiempos de desarrollo Máx., Mín., y total. Escribe los datos del plan en la tabla Resumen del Plan del Proyecto. Anota el tiempo de planificación en el Cuaderno de Registro de Tiempos.
2	Diseño	Diseña el programa. Anota el diseño en el formato especificado. Anota el tiempo de diseño en el Cuaderno de Registro de Tiempos.
3	Codificación	Implementa el diseño. Utiliza un formato estándar para introducir el código. Anota el tiempo de codificación en el Cuaderno de Registro de Tiempos.
4	Compilación	Compila el programa. Corrige todos los errores encontrados. Anota el tiempo de compilación en el Cuaderno de Registro de Tiempos.
5	Pruebas	Prueba el programa. Corrige todos los errores encontrados. Anota el tiempo de pruebas en el Cuaderno de Registro de Tiempos.
6	Postmortem	Completa la tabla de Resumen del Plan del Proyecto con los datos de tiempo y tamaño reales. Anota el tiempo postmortem en el Cuaderno de Registro de Tiempos.
	Criterios de salida	Programa probado a fondo. Diseño adecuadamente documentado. Listado completo del programa. Resumen del Plan del Proyecto completado. Cuaderno de Registro de Tiempos completado.

Fuente: Watts, 1995

Para planificar, primero se deben obtener los requisitos del proyecto y completar la tabla ´´Resumen del Plan´´ (Figura 10) del Proyecto. Finalmente, escribe el tiempo dedicado a hacer esta planificación en el ´´Cuaderno de Registro de Tiempos´´.

Figura 10. Resumen del plan del proyecto del PSP



Fuente: Watts, 1995

Tabla 6. Instrucciones del resumen del plan del proyecto del PSP

Propósito	Esta tabla trata los datos estimados y reales de los proyectos de una forma cómoda y fácilmente recuperable.
Cabecera	Introduce los siguientes datos: • Tu nombre y fecha de hoy. • Nombre y número de programa. • Nombre del profesor. • El lenguaje que utilizarás para escribir el programa.
Minutos/LOC	Antes de iniciar el desarrollo • Escribe los Minutos/LOC planificados para este proyecto. *Utiliza la velocidad Hasta la Fecha de un programa reciente del Cuaderno de Trabajos o del Resumen del Plan del Proyecto.* Después del desarrollo • Divide el tiempo total de desarrollo por el tamaño real del programa para obtener los Minutos/LOC reales. • Por ejemplo, si el proyecto se realizo en 196 minutos e hiciste 29 LOC, los Minutos/LOC serían 196/29 = 6,76.
LOC/Hora	Antes de iniciar el desarrollo: • Calcula las LOC por hora planificada para este programa dividiendo 60 por los Minutos/LOC de la casilla de Plan. Después del desarrollo • Para LOC/Hora Real divide 60 por Minutos/LOC Reales. • Para los 6,76 Minutos/LOC Reales, tenemos 60/6,76 = 8,88 LOC/Hora Reales
Tamaño Programa (LOC)	Antes de iniciar el desarrollo • Escribe bajo la columna plan, el valor estimado de Total Nuevas & Cambiadas, Máximo y Mínimo. Después del desarrollo: • Cuenta y escribe las LOC Nuevas & Cambiadas Reales. • *Para la columna Hasta la Fecha, añade LOC Reales Nuevas & Cambiadas a las LOC Hasta la Fecha Nuevas & Cambiadas de programas anteriores.*
Tiempo por fase Plan	Para el tiempo total de desarrollo, multiplica el valor de las LOC Total Nuevas & Cambiadas por los Minutos/LOC. Para el tiempo Máximo, multiplica el tamaño Máximo por los Minutos/LOC. Para el tiempo Mínimo, multiplica el tamaño Mínimo por los Minutos/LOC. *Del Resumen del Plan del Proyecto de un programa reciente, busca los valores de % Hasta la Fecha para cada fase. Utilizando el % Hasta la Fecha de programas anteriores calcula el tiempo planificado para cada fase.*
Real	Una vez acabado el trabajo, anota el tiempo real en minutos que has gastado *en cada fase del desarrollo.* Obtén estos datos del Cuaderno de Tiempos.
Hasta la fecha	*Para cada fase, escribe la suma del tiempo real y el tiempo Hasta la Fecha de los programas más recientes.*
% hasta la fecha	*Para cada fase, escribe 100 multiplicado por el tiempo Hasta la Fecha y lo divides por el total del tiempo Hasta la Fecha.*

Fuente: Watts, 1995

Antes de iniciar un proyecto, se debe completar la parte de Plan de la tabla Resumen del Plan Proyecto. Ahora, la única diferencia es que necesitaremos estimar el tiempo que vamos a dedicar a cada fase. Esto se hace asignando a cada fase, un porcentaje del tiempo de desarrollo total, basándose en la utilización del tiempo en proyectos anteriores. Para proyectos sucesivos, sin embargo, se puede utilizar los datos de proyectos anteriores para estimar el tiempo para cada fase del nuevo proyecto. Esta es la razón de por qué aparecen los valores de % Hasta la Fecha en la tabla Resumen del Plan del Proyecto.

Las columnas de Hasta la Fecha y % Hasta la Fecha de la tabla Resumen del Plan del Proyecto, proporcionan una forma sencilla de calcular la distribución de los porcentajes del tiempo de desarrollo para las fases del proceso. La columna Hasta la Fecha contiene el total de todos los tiempos dedicados en cada fase para todos los programas acabados. La columna de % Hasta la Fecha tiene el porcentaje de los tiempos de la columna Hasta la Fecha (Watts, 1995).

18.1 LA GESTIÓN DEL TIEMPO

Para practicar la gestión del tiempo, el primer paso es entender cómo utilizar el tiempo ahora. Esto se hace en varios pasos:

Clasifica las principales actividades: Cuando comenzamos a controlar el tiempo, probablemente encontraremos que gran parte del mismo lo dedicas a relativamente pocas actividades. Esto es normal. Para hacer algo, debemos centramos en pocas cosas que sean muy importantes. Si distribuyes tu tiempo entre muchas cosas, será difícil encontrarle sentido a los datos. De tres a cinco categorías deberán ser suficientes para controlar el tiempo durante el curso.

Registra el tiempo dedicado a cada una de las actividades principales: Se necesita bastante disciplina personal para registrar el tiempo de forma consistente. Debemos tomar un registro exacto, registrar el tiempo de inicio y fin de cada actividad principal.

Registra el tiempo de forma normalizada: Normalizar los registros de tiempo es necesario porque el volumen de datos aumentará rápidamente. Si no registramos y almacenamos cuidadosamente estos datos, se perderán o estarán desorganizados. Los datos confundidos o desordenados son difíciles de encontrar o interpretar. Si no intentamos tratar los datos de forma adecuada, puede que no los reunamos bien.

Guarda los datos de tiempo en un lugar adecuado: Puesto que necesitaremos guardar los registros de tiempo con los trabajos del proyecto, se deben guardar en un lugar adecuado. Esta es una de las principales utilidades del cuaderno de ingeniería (Watts, 1995).

18.2 EL CUADERNO DE INGENIERÍA

La elaboración de un cuaderno de ingeniería es tomado del libro de Humphrey del cual, hemos realizado toda la investigación con respecto al proceso de desarrollo software.

Un cuaderno de ingeniería lo utilizaremos para controlar el tiempo. También para otras cosas, tales como, guardar los ejercicios, controlar compromisos, tomar notas de clase y como un cuaderno de trabajo para anotar ideas de diseño y cálculos.

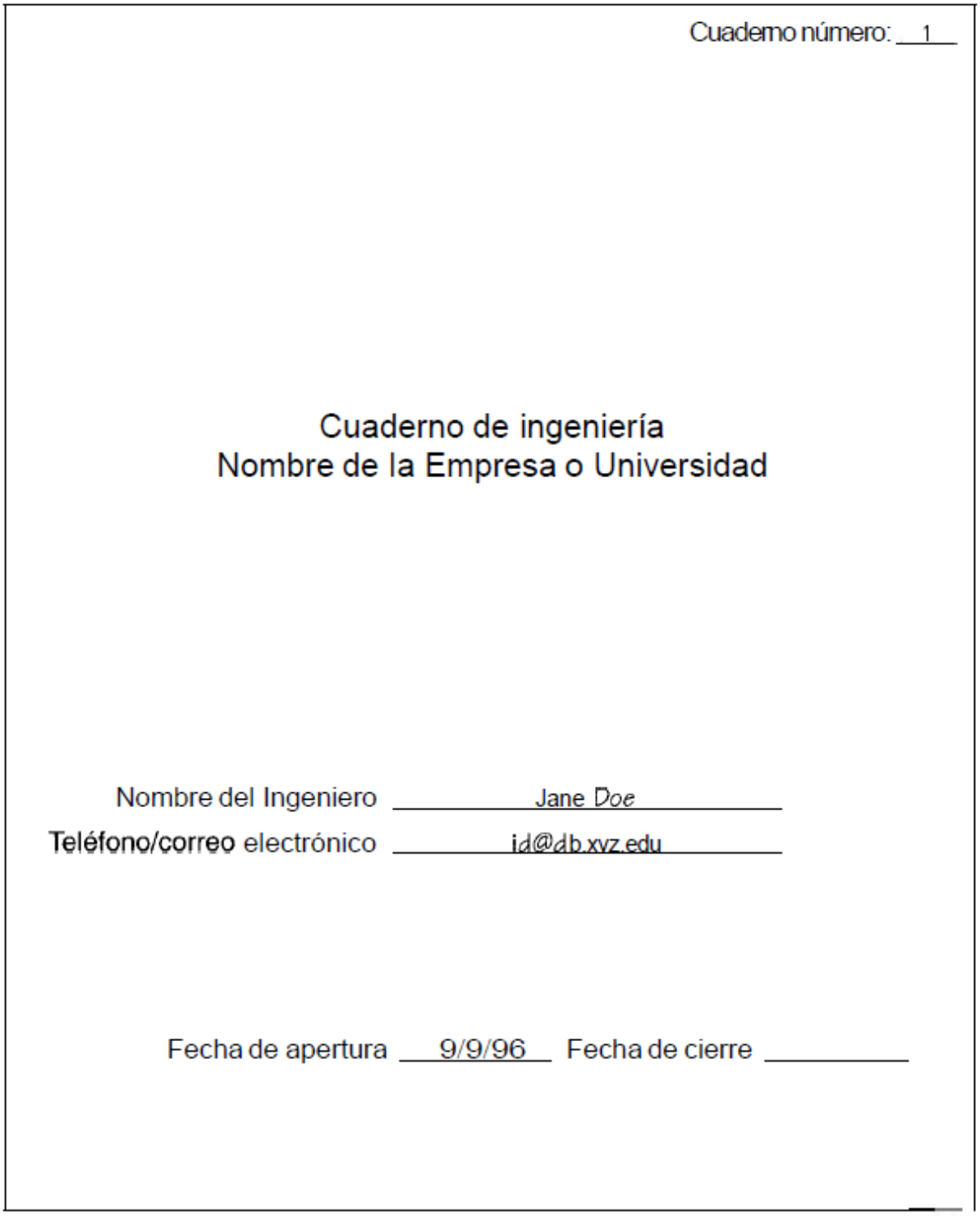
Podremos utilizarlo como una evidencia de lo que hacemos en la práctica de la ingeniería, evidencia importante para la defensa de nuestra compañía, si es que tenemos que defender la responsabilidad legal de un producto.

Una utilización adicional del cuaderno de ingeniería es la protección de los activos intelectuales de los empleados, por ejemplo, registrando ideas que se puedan patentar.

18.3 EL DISEÑO DEL CUADERNO

Primero, la portada que se muestra en la siguiente figura, es una sugerencia.

Figura 11. Ejemplo de portada del cuaderno de ingeniería



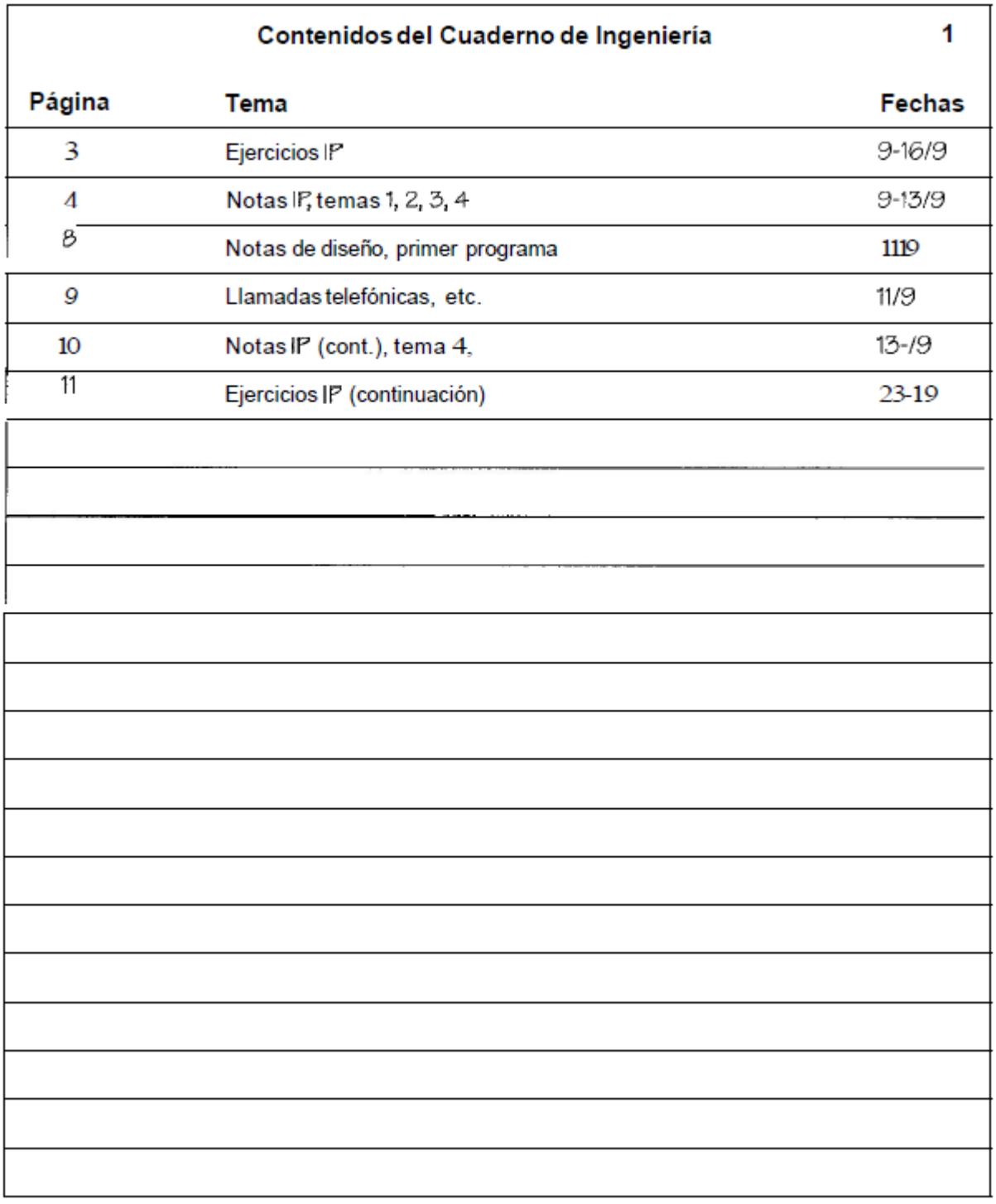
Fuente: Watts, 1995

Es recomendable etiquetar cada cuaderno con tu nombre y número de teléfono o dirección de correo electrónico. Se debe escribir la fecha de inicio de introducción de datos en el cuaderno, y cuando se haya terminado se debe escribir la fecha del último registro.

Dentro del cuaderno, se debe numerar cada página, debemos utilizar las dos primeras páginas para una breve tabla de contenidos. En los contenidos, escribimos cualquier referencia especial para que podamos encontrarla posteriormente.

Veamos un ejemplo tomado del libro de Humphrey. Un ejemplo de la página de contenidos del cuaderno de ingeniería se muestra en la figura 12. Para materias que necesitarás en el futuro, escribe a la izquierda el número de la página del cuaderno con una breve descripción del tema. Por ejemplo, el estudiante registra en la página 3 todos los ejercicios de la asignatura de IP (Introducción a la Programación) para dos semanas. Los contenidos también muestran que los ejercicios se continuarán registrando en la página 11. Un ejemplo de la página 3 del cuaderno se muestra en la figura 13.

Figura 12. Ejemplo de la página de contenidos del cuaderno de ingeniería

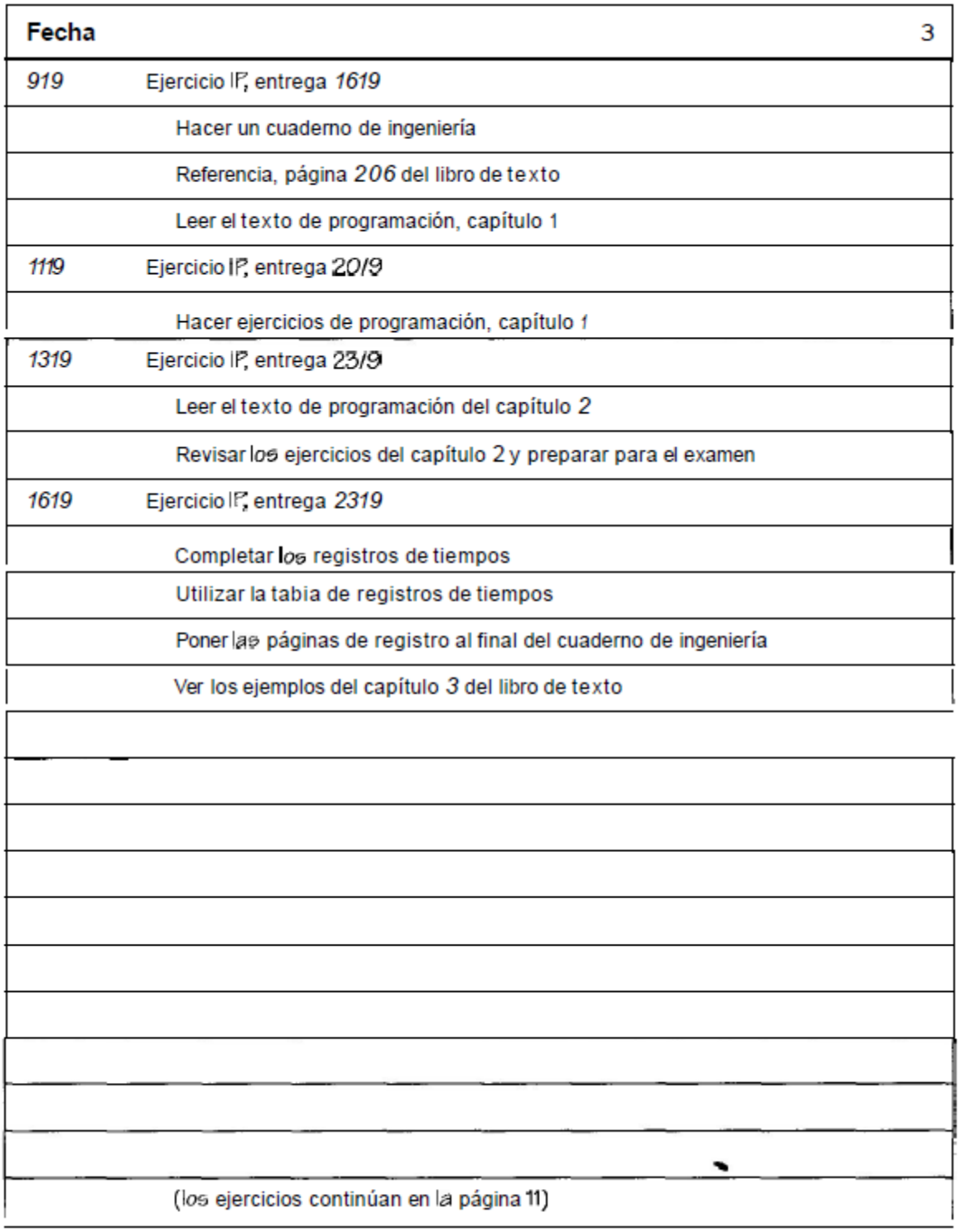


Contenidos del Cuaderno de Ingeniería 1

Página	Tema	Fechas
3	Ejercicios IP	9-16/9
4	Notas IP, temas 1, 2, 3, 4	9-13/9
8	Notas de diseño, primer programa	1119
9	Llamadas telefónicas, etc.	11/9
10	Notas IP (cont.), tema 4,	13-/9
11	Ejercicios IP (continuación)	23-19

Fuente: Watts, 1995

Figura 13. Ejemplo de una página del cuaderno de ingeniería



Fecha		3
919	Ejercicio IP, entrega 1619	
	Hacer un cuaderno de ingeniería	
	Referencia, página 206 del libro de texto	
	Leer el texto de programación, capítulo 1	
1119	Ejercicio IP, entrega 2019	
	Hacer ejercicios de programación, capítulo 1	
1319	Ejercicio IP, entrega 2319	
	Leer el texto de programación del capítulo 2	
	Revisar los ejercicios del capítulo 2 y preparar para el examen	
1619	Ejercicio IP, entrega 2319	
	Completar los registros de tiempos	
	Utilizar la tabia de registros de tiempos	
	Poner las páginas de registro al final del cuaderno de ingeniería	
	Ver los ejemplos del capítulo 3 del libro de texto	
	(los ejercicios continúan en la página 11)	

Fuente: Watts, 1995

Los contenidos también muestran que entre el 9/9 y el 13/9, el estudiante tomó notas de clase en las páginas 4, 5, 6 y 7. Después, continuó tomando notas en la página 10. Siempre que tengas que saltar algunas páginas debido a otras anotaciones, es una buena idea escribir en la parte inferior de la página dónde continúa ese tema. Véase, por ejemplo, la última línea de la figura 13.

18.4 LA ESTIMACIÓN DEL RENDIMIENTO DEFINITIVO

> Aunque nunca se puede saber con certeza el rendimiento cuando se acaba una fase, la medida de rendimiento ayudará a evaluar y mejorar el proceso. Por ejemplo, cuando encuentras 17 defectos en la revisión de código, 2 en la compilación y 1 en las pruebas, puedes asegurar de forma razonable que el rendimiento de la revisión estuvo próximo al 85%.
>
> Si se encuentra 17 defectos en la revisión, otros 15 en la compilación y 8 más en las pruebas, el rendimiento inicial de la revisión es del 42,5% y probablemente disminuirá en el futuro. Cuando encuentres 23 defectos en la compilación y pruebas, puedes asegurar de forma razonable que aún hay unos pocos defectos más en el producto.
>
> Debido a que durante los años de utilización es posible que no se encuentren todos los defectos, nunca se tendrá la certeza. Si el número de defectos encontrados disminuye bruscamente con cada eliminación de defectos, los valores del rendimiento son probablemente bastante precisos. Después de que hayamos reunido datos de defectos, se puede desarrollar medidas de rendimiento para cada fase. Entonces podremos calcular el número de defectos que quedan en cualquier producto que se desarrolle.
>
> Una regla de utilidad empírica es asumir que los defectos que quedan en un producto equivalen al número de los mismos encontrados en la última fase de eliminación (Watts, 1995)

Veamos un ejemplo tomado del libro de Watts S. Humphrey:

> Considera de nuevo el caso con los 17 defectos encontrados en la revisión de código, 2 en la compilación y 1 en las pruebas, la estimación actual del rendimiento de revisión es de 17/(17 + 2 + 1) = 85%. La regla empírica asumiría que se encontrará un defecto más. Esto daría un rendimiento final de 17/(17 + 2 + 1 + 1) = 80,95%. En el otro ejemplo, 17 defectos encontrados en la revisión de código, 15 en la compilación y 8 en las pruebas. Aquí, el valor del rendimiento actual es de 17/(17 + 15 + 8) = 42,5%. La regla de utilidad empírica sugiere que se encontrarán 8 defectos más, dando un rendimiento estimado definitivo de revisión de 17/(17 + 15 + 8 + 8) = 35,4% (Watts, 1995)

18.5 LOS BENEFICIOS DE UN RENDIMIENTO DE PROCESO DEL 100%

> El objetivo de la revisión de código sería alcanzar de forma consistente un rendimiento de proceso del 100%. Si se puede lograr, encontraremos todos los defectos y no dejaríamos ninguno para encontrarlo en la fase de compilación o pruebas. Esto no solamente ahorrará tiempo de compilación y pruebas, sino que el mayor beneficio estará en los costes y planificación del proyecto. Con módulos de programas libres de defectos, el equipo de desarrollo no dedicaría tiempo a probar los programas. Esto ahorraría los costes de las pruebas, reduciría el tiempo de las mismas y mejoraría las planificaciones del desarrollo. También produciría mejores productos.
>
> Lograr un rendimiento de un 100% no es fácil. Requiere tiempo, práctica y un gran acopio de datos y análisis. El desarrollo de software de alto rendimiento, es una habilidad que puede aprenderse y mejorarse (Watts, 1995).

Las siguientes son palabras de Watts S. Humphrey acerca del rendimiento en cada ingeniero a la hora de desarrollar.

> El trabajo de realizar software de calidad es muy parecido a tocar un instrumento musical. Hasta que tú puedas hacer un código que tenga pocos o ningún defecto, no lograrás ser un ingeniero software. Mientras dediques la mayor parte de tu tiempo a encontrar y corregir defectos, no estarás actuando como un profesional. Con métodos de calidad disciplinados, sin embargo, harás pequeños programas libres de defectos de una forma coherente. Entonces, podrás centrarte en desafíos más importantes para producir grandes programas de alta calidad. Aunque dominar estas disciplinas requiere tiempo y un esfuerzo constante, verás que mejoras gradualmente. Su objetivo personal, sin embargo, debe ser alcanzar un rendimiento del 100% (Watts, 1995).

18.6 CÓMO MEJORAR LAS TASAS DE REVISIÓN

> No es de preocupar las tasas de revisión de código defecto/hora hasta que de forma regular se encuentren casi todos los defectos antes de la compilación y las pruebas. Se Continúa comprobando los datos de defectos para ver qué pasos de la revisión deberían haber encontrado los defectos que se encontraron en la fase de compilación y en la de pruebas. También, continuamos actualizando tu lista personal de comprobación para la revisión de código para encontrar estos defectos.
> Una vez que encontremos la mayor parte de los defectos en la revisión de código, debemos pensar cómo mejorar las tasas de revisión de código (Watts, 1995).

Para hacer esto, se debe identificar cualquier paso de la revisión que se allá pasado por alto.

A continuación, tenemos que reconsiderar las razones para incluir estos pasos (vistos en el capítulo ‘’Una estrategia para la eliminación de defectos”) en primer lugar. Si estas cuestiones ya no son un problema, nos saltamos los pasos.

Por otra parte, si vemos que estas pruebas son aún importantes, combinamos varias de ellas para poder hacerlas más rápidamente. Con cada programa, continuamos controlando los datos de defectos y sustituimos cualquier paso de la revisión que podría haber detectado defectos pasados por alto posteriormente. Cuando los pasos no son efectivos, no podemos dudar en eliminarlos.

18.7 CÁLCULO DEL VERDADERO COSTE DE CALIDAD (CDC)

Para calcular los verdaderos costes de fallos y valoración, se debe dividir los tiempos de revisión, compilación y pruebas entre los componentes respectivos de valoración y fallos. Por ejemplo, podemos denominar el tiempo de compilación en el que no se encuentran defectos de compilación (valoración) o C, y al tiempo de corrección de defectos durante la compilación como tiempo de compilación (fallos) o C1. Así, C1 + Cv = C es el tiempo total de compilación. Para los tiempos de revisión y pruebas, tenemos R1 + Rv = R como tiempo total de revisión y P1 + Pv = P como tiempo total de pruebas.

Utilizando estos parámetros, podemos calcular con precisión la valoración y fallos CDC de la siguiente manera:

Valoración CDC = 100*(Rv + Cv + Pv) / (tiempo total de desarrollo)

Fallos CDC = 100*(Rf + Cf + Pf)/ (tiempo total de desarrollo)

Para calcular Rv, primero se calcula R1 a partir del Cuaderno de Registros de Defectos.

Puesto que los valores de V/F y CDC son más precisos y bastante sensibles a los tiempos de reparación de defectos, no es viable utilizar este método a menos que se esté midiendo los tiempos de corrección con cronómetro [22]

18.8 COMO HACER UN COMPROMISO CON LA CALIDAD

‘’Muchos ingenieros del software ven los defectos como bugs. Los tratan como algo que arrastran desde alguna parte, pero no es algo de lo que se sientan responsables. Es en definitiva es un error. Los defectos son introducidos por los mismos ingenieros y cuando un programa tiene defectos es defectuoso. ’’ [22]

Cuando los ingenieros están comprometidos con la calidad, prestan atención y cuando prestan atención, son muy cuidadosos con su trabajo. Cuando son muy cuidadosos, producen mejores productos.

> El primer paso para producir un software de calidad es fijar que la calidad es lo más importante. El segundo paso es establecer el objetivo de producir programas libres de defectos. Para tener oportunidad de cumplir esta meta, debemos medir la calidad de los programas y dar los pasos que mejoren dicha calidad.
>
> Como seres humanos, todos introducimos defectos ocasionalmente. El reto es gestionar nuestra falibilidad. Esto, sin embargo, es una lucha interminable. Los productos software del futuro serán más sofisticados y más complejos. Nos enfrentaremos a problemas que están constantemente cambiando. La cuestión no es si puedes hacer un trabajo libre de defectos, sino, si somos lo bastante cuidadosos para continuar haciéndolo. Esto no solamente hará que el ingeniero y la organización sean más productivos, sino que también tengamos más satisfacción por el trabajo realizado.

19. MODELO DE DESARROLLO DE EQUIPOS DE TUKMAN

Hemos querido realizar una especie de ‘’plus’’ a nuestro trabajo, con referente al manejo de equipos cuando queremos desarrollar un proyecto no sólo software, sino también, cualquier proyecto administrativo. Este modelo fue hecho por Bruce Tuckman, un experto en el campo de la psicología.

El modelo de Tuckman explica que mientras el equipo desarrolla madurez, habilidades y establece relaciones entre sus miembros, el líder va cambiando su estilo de liderazgo. Comenzando con un estilo directivo, moviéndose hacia el coaching, luego participando y finalizando con delegación casi independiente. La autoridad y libertad extendidas por el líder hacia el equipo van en aumento conforme el control del líder va en disminución.

Las fases de este proceso son:

19.1 FASE 1: FORMACION

Se caracteriza usualmente porque la gente trata de destacarse, asimismo se denota inseguridad y deficiencia entre sus miembros, a pesar de que los integrantes extrovertidos rápidamente asumirán alguna clase de liderazgo. Predomina la conciencia de mantenerse.

Hay una alta dependencia en el líder en cuanto a guía y dirección. Roles individuales y responsabilidades son poco claras. El líder se debe de preparar para responder gran cantidad de preguntas acerca del propósito del equipo, objetivos y sobre relaciones externas. Los procesos son casi siempre ignorados. Los miembros evalúan la tolerancia del sistema y del líder. El líder dirige.

19.2 FASE 2: CONFLICTO

Ya establecido el grupo, éste es un periodo de bromas para alcanzar una posición, autoridad e influencia entre sus miembros. Se hace referencia de esta fase como de Insatisfacción.

Este es un período de probando al líder. Aparecen o se generan desacuerdos y los roles son eventualmente repartidos. Los líderes que salieron al inicio probablemente no sobrevivirán éste período: es la fase más incómoda de la vida del grupo, es como la adolescencia del grupo.

El equipo debe mantenerse enfocado en sus metas para evitar que sobrevengan distractores causados por las relaciones entres los miembros y por asuntos emocionales. Compromiso es requerido para poder progresar. El líder actúa como coach.

19.3 FASE 3: NORMALIZACIÓN

Qué clase de comportamiento y contribución es aceptable y cual no. Los miembros exploran detrás del proceso de lucha de poder y comienzan a formular alguna idea sobre la identidad de grupo, ahora el grupo está en la mente de sus miembros. Desde luego esto es raro que se haga explícitamente, por lo que se puede retroceder a la fase anterior de Conflicto.

Se forma acuerdo y consenso dentro del equipo, el cual responde bien a la enseñanza del líder. Roles y responsabilidades son claras y aceptadas. Grandes decisiones son tomadas por acuerdo del grupo; las pequeñas decisiones son delegadas individualmente o pequeños equipos que se forman dentro del grupo. La unión y el compromiso son fuertes.

El equipo se puede involucrar en actividades divertidas y sociales. El equipo discute y desarrolla sus procesos y su forma de trabajo. El líder es respetado por

el equipo y parte del liderazgo es compartido por el equipo. El líder facilita y capacita.

19.4 FASE 4: DESEMPEÑO

A partir de este momento, el grupo comienza a hacer su trabajo sobre la base de una estructura relativamente estable.

El equipo está estratégicamente alerta. El equipo sabe de antemano por qué lo está haciendo y para qué lo está haciendo. El equipo tiene una visión compartida y está capacitado para pararse por su propia cuenta sin la intervención o participación del líder. Hay un enfoque en lograr resultados y el equipo toma decisiones por sí mismo. El equipo tiene un alto grado de autonomía.

Los desacuerdos ocurren pero ahora son resueltos positivamente dentro del equipo y los cambios necesarios al proceso y a la estructura son realizados por el equipo.
Los miembros del equipo se cuidan entre ellos. El equipo requiere que el líder delegue tareas y proyectos. El equipo no necesita ser instruido o asistido. Los miembros del equipo pueden pedir asistencia del líder. El líder delega y ve sobre todos.

19.5 FASE 5: TERMINACION

La etapa de Terminación es sin lugar a dudas muy relevante para los integrantes del grupo y su bienestar, pero no hacia la principal tarea de administrar y desarrollar un equipo, que claramente se centra en las cuatro fases originales.

La etapa de desintegración es cada vez más frecuente con el surgimiento de las organizaciones virtuales, las cuales se terminan cuando se logra el proyecto o fin buscado.

20. RECOMENDACIONES

La compañía es una de las empresas líderes en el mercado del software concentrada en mejorar de forma constante sus procesos, por este motivo debería ajustar los siguientes procesos:

- **En el proceso de Elicitación**: Antes que el gerente encargado de la elicitación tenga la primer cita con el cliente, este debe tener bien claro que hace el cliente y cuál es su mercado objetivo, para que de esta forma, cuando se esté elicitando, sea más fácil para éste, identificar los puntos críticos para el cliente, identificar zonas críticas para el desarrollo y los usuarios a los cuales estará concentrado el desarrollo.

- **En el proceso de Análisis**: Identificar de forma temprana que tipo de personas debe hacer parte del proyecto, tomando en cuenta el conocimiento, comunicación, integración con el equipo. De igual forma en este punto se deben de identificar los roles que cada uno de los integrantes va a cubrir. Incluir en las citas con el cliente al equipo de desarrollo, o si esto no es posible, separar momento para contextualizar al equipo y este tenga conocimiento global sobre la aplicación a desarrollar.

- **En el proceso de desarrollo**: Capacitar si es necesario al equipo de trabajo en la medición y toma de tiempos, para futuros análisis. Documentar lecciones aprendidas, esto para evitar que se cometan errores que ya han tenido solución en el proyecto, he incluso puede ser de valor para otros proyectos. Generar repositorios de conocimiento, esto es dado el caso un integrante del equipo deba salir, el conocimiento de este permanezca dentro del proyecto y dado el caso que entre un nuevo integrante al equipo, este tenga una base en la filosofía que se maneja en el proyecto.

- **En el proceso de pruebas**: Desde el análisis definir qué tipos de pruebas se deben llevar a cabo durante el proceso de desarrollo. Capacitar al desarrollador para que este pruebe de forma adecuada los desarrollos. Aunque inicialmente es difícil identificar los errores, es importante que en el momento que se detecte un error en los desarrollos, que el equipo de trabajo esté dispuesto a mejorar la calidad del producto antes de que este sea liberado (esto con el fin de evitar costos adicionales y atrasos en el desarrollo). Para poder lograr esto es recomendable, realizar inspecciones de todas las interfaces de módulos y sus interacciones, inspeccionar los requisitos para asegurar que todas las funciones importantes son adecuadas para el desarrollo.

Si realmente se desea lograr reducir los costos en los desarrollos, es importante implantar una filosofía de aprendizaje continuo, en donde el personal esté capacitado en nuevas tecnologías y en nuevos modelos de desarrollo.

Capacitar a los gerentes de proyecto en estrategias de comunicación e integración de equipos de trabajo, esto tomando en cuenta que es muy malo para un proyecto que un equipo este desacoplado, ya que se puede perder la comunicación entre los integrantes del mismo, causando que el proyecto falle.

La compañía deberá establecer mecanismos en la toma de tiempos, tales como PSP, para los ingenieros desarrolladores, con el fin de mejorar de forma continua la calidad de los productos que se estén o que estén próximos a ser desarrollados, tomando en cuenta que esta es una base firme para realizar proyecciones a futuro relacionadas al cuanto tiempo tardarán en desarrollar los productos.

21. CONCLUSIONES

- Los procesos bien definidos y, que son constantemente evaluados según cambia el entorno organizacional, traerá a la organización en corto y mediano plazo, excelentes resultados a la organización

- La evaluación de procesos permite concluir de forma cualitativa y cuantitativa la calidad de los diferentes tipos de procesos que existen en la compañía, que en nuestro caso, fueron los procesos con referencia al desarrollo software.

- Las evaluaciones deben llevarse a cabo como un proyecto organizado, cuya magnitud varía según las condiciones particulares de la compañía. Para que la evaluación de procesos se realice adecuadamente se debe efectuar un proceso de planeación inicial, en el cual se deben definir claramente los participantes del proyecto y sus responsabilidades, así como los lineamientos generales que regirán cada una de las actividades, tales como objetivos, estándares, documentos de referencia, indicadores, etc.

- Las prácticas base que ofrece el modelo de evaluación de procesos de ISO/IEC 15504 permite construir metodologías de evaluación gracias a la clara descripción que ofrecen de los procesos de software.

- La evaluación de los procesos utilizando ISO/IEC 15504 se convierte en una herramienta de gran importancia que debe ser utilizada en la compañia con el fin de determinar la capacidad de sus procesos y a partir de estas mediciones, trabajar en el mejoramiento continuo de estos.

- Tomando en cuenta que la compañía tiene un enfoque de mercado, definido a dar soporte sobre aplicaciones ya existentes y realiza desarrollos nuevos de software, no es conveniente adoptar prácticas de Modelos formales, ya que estos imprimen gran robustez al proceso de elicitación y análisis. Adoptar estas

metodologías en el momento en el que el cliente indique que el software a dar soporte esté desarrollado con una estrategia de metodologías formales y éste insista en que se debe mantener el modelo con el que viene el proyecto.

- Para el caso particular de la compañía, la metodología de SCRUM se acoge mejor al negocio debido a que, con esta metodología se maneja las cosas de forma iterativa. Lo que indica que el cliente puede fijar sus expectativas e iniciar a darle valor al producto incluso desde antes de que este finalice. De igual forma el cliente puede redirigir el proyecto de acuerdo a las prioridades y de acuerdo a los cambios que se vean en el mercado.

- La metodología SCRUM a su vez le da control al equipo de trabajo, de tal manera que pueden trabajar de forma más eficiente y que estos pueden cumplir a cabalidad los objetivos planteados durante el proceso de desarrollo del proyecto, es de aclarar que todo proyecto que esté desarrollado con esta metodología esta dado por un número de iteraciones y que cada una de estas iteraciones no son liberadas hasta que estas cumplan los estándares de calidad predefinidos por el cliente.

- Tomando en cuenta que cada vez se busca aumentar la calidad del producto y la eficiencia para el desarrollo de los productos, es de gran valor para la compañía incluir dentro de las capacitaciones del personal, el modelo PSP, ya que este favorece al ingeniero dándole las herramientas necesarias para medir su trabajo y aumentar cada vez más la calidad de los productos que desarrolla basado en las experiencias de desarrollos previos.

- En conjunto con el modelo PSP, es de gran valor para la compañía implantar el modelo TSP, tomando en cuenta que estos dos modelos se complementan de tal forma que fomenten la mejora continua y facilitan la generación de productos de calidad.

- En el momento en el que se lleva a cabo el proceso de desarrollo, es importante prestarle atención a las pruebas, ya que estas a la larga, son las que indican si el producto que se esta desarrollando es de calidad o no. De igual forma es importante tener bien establecido al interior de la compañía un modelo de pruebas reutilizable y escalable, ya que esto forma una base de conocimiento la cual facilitará el desarrollo de nuevas pruebas en diferentes productos.

- Es claro que los procesos de la compañía deben de mejorar tomando en cuenta que el enfoque de mercado que se tiene es hacia desarrollos nuevos y soporte de aplicaciones. En esto se evidencia que existen partes de los procesos que no se han aprovechado o no se han tomado en cuenta ya que no se acogen a los proyectos nuevos.

- Para el rol de desarrollador, se debe tener el conocimiento y los criterios necesarios para poder aceptar o no el desarrollo de un nuevo producto o servicio, ya que no es una buena práctica incluir dentro de una iteración ya definida, desarrollos que no han sido gestionados apropiadamente.

- La gerencia debe de estar dispuesta a ayudar y escuchar al equipo de desarrollo con el fin de mitigar conflictos no solo con el cliente, sino al interior del equipo de trabajo. De igual forma es de importancia que la gerencia busque espacios extra laborales enfocados a la integración del equipo, con el fin de mejorar la comunicación entre el equipo de trabajo y demás empleados dentro de la organización.

BIBLIOGRAFIA

- ABRAHAMSSON, Pekka. WARSTA, Juhani. SIPONEN, Mikko y RONKAINEN, Jussi. New directions on agile methods a comparative analysis. Portland: In Proceedings of the 25th international Conference on Software Engineering, 2003. 254 p.

- ACM 2008. Proceedings of the 46th Annual Southeast Regional Conference. Alabama, 2008. 548 p.

- AMARAL, Luis. A gap analysis methodology for the TSP. Porto: Quatic 2010, 2010. 429 p.

- CONBOY, K y FITZGERALD, B. Towards a conceptual framework of agile methods: A study of agility in different disciplines. Newport Beach: ACM workshop on Interdisciplinary software engineering research, 2004.

- CRISTIÁ, Maximiliano. Introducción al testing de software. Universidad nacional de Rosario, 2009. 8 p.

- DURAN TORO, Amador y BERNARDEZ JIMENEZ, Beatriz. Metodología para la elicitación de requisitos de sistemas software. Universidad de Sevilla, 2000. 72 p.

- Formal Methods. Southwest Baptist University. New York (Citada 5, 6, 10 de Octubre) http://www.afm.sbu.ac.uk/

- FULLER, David. Roles en el desarrollo de software. Apuntes de ingeniería de software Vol. 1.3, 2003. 26 p.

- GALÁN MORILLO, Francisco José y CAÑETE VALDEÓN, José Miguel. Métodos formales orientados a objetos. Sevilla: ETSI Informática, 2006. 20 p.

- GRACK, Garry y ROBINSON Kyle. Integrating improvement initiatives: Connecting six sigma for software, CMMI, PSP and TSP. 2003. 5 p.

- LEVESON, Nancy. Safe ware, System safety and computers. Reading, MA: Addison-Wesley. 1995.

- LOPEZ TRUJILLO, Yucely; ANDRÉ AMPUERO, Margarita e INFANTE ABREU, Ana lilian. Formación de roles y buenas prácticas en el trabajo por la calidad del ingeniero informático. En: Revista chilena de ingeniería vol. 19, 2011. 14 p.

- MANNA, Zohar y PNUELI, Amir. The temporal logic of reactive and concurrent systems. Springer-Verlag, 1991.

- MENDELSON, Elliot. Introduction to mathematical logic. Wadsworth & Books/Cole, 1987. 440 p.

- OCAMPO HENAO, Ana Catalina y RUIZ URIBE, Juan Diego. Evaluación de los procesos de software utilizando como referencia el estándar ISO/IEC 15504. Medellín: Universidad EAFIT, 2008. 174 p.

- RATIONAL THE SOFTWARE DEVELOPMENT COMPANY. Rational Unified Process, Best Practices for Software Development Teams. Dual Headquarters: Rational software. Lexington, MA. 2011. 18 p.

- REYES, Raúl. La importancia de los procesos. En: Artículos inteligentes que permiten tener un ángulo diferente de pensamiento. Monterrey 2009, Vol. 1 (Citada 17 septiembre 2012) http://smart-text.blogspot.com/2009/10/la-importancia-de-los-procesos.html

- SCHWABER, Ken y SUTHERLAND Jeff. La guía de Scrum. La guía definitiva de scrum: Las reglas del juego, 2011. 17 p.

- UTTING, Mark y LEGEARD, Bruno. Practical Model - Based testing: A tool approach. Morgan Kaufmann Publishers Inc.: San Francisco, 2006.

- VAN LOON, Han. Process Assesment and ISO/IEC 15504. A reference book. Second edition. Springer.

- VELÁSQUEZ, Jhon. Fundamentación de la gestión por procesos módulo 1. Medellín: Universidad EAFIT, 2012. 99 p.

- WATTS, Humphrey y OVER, James. Leadership, Teamwork, and Trust. Reading, MA: Addison-Wesley. 2010, 307 p.

- WATTS, Humphrey. A discipline for Software Engineering. Reading, MA: Addison-Wesley, 1995.

- WATTS, Humphrey. Introducción al proceso software personal. Carnegie Mellon University. 1998, 322p

- WATTS, Humphrey. Managing the software process. Reading, MA: Addison-Wesley. 1995.

- WIKIPEDIA. ISO/IEC 15504. Última edición: 8 Agosto 2012 (Citada 20 septiembre 2012). http://es.wikipedia.org/wiki/ISO/IEC_15504.

- ZHANG, Tao y LEE, Byungjeong. Complementary Classification techniques based Personalized Software Requirements Retrieval with Semantic Ontology and User Feedback. IEEE International conference on Computer and information Technology, 2010. 6 p.