ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

"PROPUESTA METODOLÓGICA PARA ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE EN PROYECTOS PEQUEÑOS Y MEDIANOS ORIENTADOS A OBJETOS"

DANILO GEOVANNY BARRENO NARANJO

Tesis presentada ante la Escuela de Postgrado y Educación Continua de la ESPOCH, como requisito para la obtención del Título de Magíster en Informática Aplicada

RIOBAMBA – ECUADOR

2009

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

CERTIFICACIÓN:

EL TRIBUNAL DE TESIS CERTIFICA QUE:

El trabajo de investigación titulado "PROPUESTA METODOLÓGICA PARA ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE EN PROYECTOS PEQUEÑOS Y MEDIANOS ORIENTADOS A OBJETOS", de responsabilidad del Sr. Danilo Geovanny Barreno Naranjo ha sido prolijamente revisado y se autoriza su presentación.

Tribunal de Tesis:

Ing. Byron Vaca _____

PRESIDENTE

Dra. Narcisa Salazar _____

DIRECTORA

Ing. Blanca Hidalgo _____

MIEMBRO

Dr. Alonso Álvarez _____

MIEMBRO

Riobamba, Octubre 2009

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

ESCUELA DE POSTGRADO Y EDUCACIÓN CONTINUA

TESIS DE GRADO

Previa a la Obtención del Título de:

MAGÍSTER EN INFORMÁTICA APLICADA

TITULO:

"PROPUESTA METODOLÓGICA PARA ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE EN PROYECTOS PEQUEÑOS Y MEDIANOS ORIENTADOS A OBJETOS"

MAESTRANTE: Danilo Geovanny Barreno Naranjo

TUTOR: Dra. M.I.A. Narcisa Salazar

DEDICATORIA

Este trabajo los dedico a mi hijo quien siempre me apoya con una sonrisa, un beso o con su simple presencia, a mi esposa quien ha soportado mi ausencia durante mucho tiempo.

AGRADECIMIENTO

Mi eterna gratitud a todos quienes me apoyaron en todo momento, especialmente a mi familia, a mi esposa y a mi hijo.

A la Escuela Superior Politécnica del Chimborazo institución que me ha ofrecido la formación de pre grado y post grado y a la Universidad Estatal de Bolívar institución que me ha brindado la oportunidad de seguir superándome.

TABLA DE CONTENIDOS

CAPÍTULO III

MARCO PROPOSITIVO

CAPÍTULO IV

MARCO HIPOTÉTICO Y METODOLÓGICO

CAPÍTULO V

ANÁLISIS E INTERPRETACIÓN DE RESULTADOS

RESUMEN

El estudio establece la realidad de los sistemas desarrollados como tesis de grado o prácticas pre profesionales, los cuales en un alto porcentaje no son usados por mucho tiempo, la principal causa para esto es que los sistemas no tienen las funciones que los usuarios desean, por tal motivo es vital crear una metodología de especificación de requisitos que sea práctica.

La metodología desarrollada es basada en Rational Unified Process (RUP), enfocándose a desarrollar sistemas que mantiene una base de datos, debido a que de 25 casos estudiados, 22 mantienen base de datos, 2 son sistemas multimedia y un sistema experto. Además permite que la especificación de requisitos de software se realice en un ambiente iterativo, mejorando de esta manera la aceptación de los sistemas.

SUMMARY

This study establish the reality of development software system in thesis or pre professional practices, A high percent of system software don’t use for a long time, the first reason is: the software system don’t have correct functions, The functions don’t be the user want, For this reason is vital to make a workable specify requirements methodology.

The development methodology is based in Rational Unified Process, the goal is to make software system that support database because of 25 studied case, 22 are software system that support database, 2 are multimedia software system and 1 is a expert system software. Moreover whit the methodology you can make specify iteratively requirements, for this way you can make more used software system.

INTRODUCCION

A pesar de que la ingeniería de software ha llegado a especificar principios, métodos y técnicas para el desarrollo de software de calidad uno de los graves problemas que tiene el desarrollo de software es el poder definir un conjunto de especificaciones de software completo y que nos lleve a lograr una alta satisfacción del cliente.

El software originalmente era creado por la misma persona que lo requería, por tal motivo no se necesitaba una especificación de requerimientos, pues el desarrollador conocía a fondo el problema a resolver y sabía lo que necesitaba en su software, esto cambio rápidamente y sin duda alguna este fue una de las causas de la denominada Crisis del Software (finales de los 60's e inicio de los 70's), ante esta necesidad se crearon diferentes técnicas que pretendieron eliminar este serio inconveniente.

Las primeras técnicas de especificación aparecen como descripción de procedimientos, a través de seudo códigos como español estructurado y los diagramas de flujo, estas técnicas persiguen describir los detalles algorítmicos de un proceso que recorrían los datos para transformase en información, estos dos métodos se utilizaron mucho en la programación estructurada y son muy útiles para describir pequeños módulos de software que realizan una tarea específica, pero cuando se trata de describir el funcionamiento de sistemas más complejos pierden objetividad, otros métodos para recolectar requerimientos de software que son muy validos son las entrevistas y

encuestas que permiten tener una visión más calara de lo que los usuarios de un sistema desean del software, el problema es que estos resultados no son fácilmente verificables, es decir como presentarlo a nuestros clientes y que ellos nos digan si lo que el desarrollador a captado es en realidad lo que ellos desean.

En cuanto a Programación Orientada a Objetos han existido diferentes forma de obtener los requisitos de software y como expresarlos dependiendo de cada autor, todos estas metodologías han confluido a una sola que es la más difundida y usada que es la del Lenguaje de Modelado Unificado (UML), que es utilizado por la metodología Proceso Unificado Racional (RUP) que especifica los requerimientos del usuario a través de los casos de uso que son diagramas que pueden ser entendidos de forma más fácil por las personas que no son del área de sistemas, esta metodología es muy útil para proyectos de software de gran tamaño, para proyectos pequeños el esfuerzo en mantener toda la documentación y control que exige RUP llega a ser prohibitivo de usarlo.

En nuestro medio los proyectos de software que se emprenden generalmente son pequeños o medianos, con equipos de desarrollo que generalmente comprenden no más de 3 personas lo que hace muy difícil aplicar eficientemente el modelo de RUP, especialmente en los trabajos de tesis de grado y de prácticas preprofesionales.

Identificándose entonces la necesidad de desarrollar un proceso investigativo que permita crear una metodología que permita realizar especificaciones de software que persigan la calidad del software en proyectos pequeños y medianos.

ANTECEDENTES

En los diferentes ciclos de vida que han existido siempre se ha establecido como una etapa imprescindible la Especificación de Requisitos de Software, en los primeros ciclos de vida este paso se ejecutaba al inicio (cascada) y con los resultados de éste se construía el sistema y cada cambio era un problema pues mientras más tarde se detectaba un error más costoso era corregirlo, pronto se construyeron ciclos de vida que permitían insertar cambios con menor impacto como el ciclo de vida en V y otros que reconocieron que la especificación de requisitos no era estática si no dinámica.

Con el aparecimiento de la teoría de programación orientada a objetos se crearon ciclos de vida iterativos, algunos basados en los anteriores ciclos de vida como el modelo evolutivo que está basado en el modelo espiral.

Los ciclos de vida y las metodologías orientadas a objetos convergen en una sola, gracias al esfuerzo de Rational Software que generó el Rational Unified Process (RUP), a pesar que este proceso es altamente adaptable no se ajusta a la realidad del desarrollo de software que se realiza en la Universidad Estatal de Bolívar, pues demanda de un grupo de al menos diez desarrolladores, cuando en los sistemas desarrollados el número máximo de participantes es de dos, por tal motivo la utilización de RUP es poco práctica.

JUSTIFICACIÓN

Las bases para un desarrollo de software exitoso son sin duda alguna los requisitos de software, por más riguroso que sea el proceso de producir los artefactos software, y por más exitosa que sean las buenas prácticas, estas se quedan sin piso si lo que van a construir no es lo que el usuario del sistema quiere, es decir se llegará a tener un muy buen producto creado de forma muy eficiente pero que hace y tiene lo que nadie quería, por tal motivo la especificación de requerimientos es la base fundamental de todo desarrollo de software.

La concepción general es que los requisitos de software se lo hace al inicio y luego de esto se los sigue rigurosamente, pues estos no deberían variar, esto sería verdad en un ambiente ideal, la verdad es que conforme se avanza el desarrollo del sistemas la mayoría de los requisitos aparecen y se van modificando continuamente, sin duda alguna esto lleva a que el costo del software se incremente y que la calidad del mismo se vea afectada principalmente cuando se tiene un límite de tiempo que esta por expirar, este es el caso real del desarrollo de tesis de grado y proyectos de prácticas preprofesionales.

El esfuerzo en definir los requisitos de software deberá estar distribuidos en todas las etapas de desarrollo del software, pero la mayoría de ellos deben estar claramente definidos en el Análisis de Requisitos y en el Análisis, en el resto del ciclo de vida se deberían hacer ajustes a estos.

El análisis de requisitos como una etapa más del desarrollo de software, esto es debido a su importancia esta etapa estaría dedicada al entendimiento del problema por parte de los desarrolladores, la etapa de análisis sería la formalización de este entendimiento y la presentación a los usuarios para llegar a acuerdos, refinarlos y dejarlos listos para proceder al diseño.

A pesar del estudio de la ingeniería de software y al conocimiento de procedimientos, buenas prácticas, notaciones, etc., no se ha podido dejar de lado la costumbre de “empezar programando”, se ve a la documentación que debe acompañar a un desarrollo de software como un trabajo adicional, muchas veces se realiza el trabajo inverso, es decir se empieza programando y luego dependiendo del resultado se realiza la documentación.

Lo que se desea es presentar una metodología con pasos definidos y con buenas prácticas de software que permita generar requisitos de usuario bien definidos y aceptados por los clientes, los cuales sirvan de bases para el desarrollo de toda la aplicación.

Se espera obtener como resultado de esta investigación una metodología para especificación de requisitos para proyectos medianos y pequeños que permita alcanzar un software que satisfaga las necesidades de los clientes y que sea usado por los mismos para sus labores diarias y no se conviertan en tan sólo un requisito para logar la graduación en la escuela de ingeniería en sistemas de la Universidad Estatal de Bolívar.

OBJETIVOS

GENERAL

Crear una metodología que permita definir requisitos de software en proyectos pequeños y medianos orientados a objetos que permita mejorar la productividad de software en los proyectos de tesis y pasantías en la Universidad Estatal de Bolívar

ESPECÍFICOS

- Estudiar las metodologías existentes para especificación de requisitos de software y evaluar su aplicación práctica en el software generado en las tesis y pasantías en la Universidad Estatal de Bolívar.
- Estudiar experiencias previas en especificación de requisitos y compararlos con los resultados obtenidos.
- Evaluar el uso que ha tenido el software generado en las tesis y pasantías en la Universidad Estatal de Bolívar.
- Analizar experiencias en la aplicación de metodologías generalmente aceptadas en proyectos pequeños y medianos.
- Definir pasos específicos a seguir para obtener los requisitos en proyectos pequeños y medianos.
- Definir las buenas prácticas que deberían aplicar en la especificación de requisitos de software en proyectos pequeños y medianos.
- Aplicar la metodología propuesta en proyectos de tesis y pasantías.
- Evaluar resultados.

CAPÍTULO I

MARCO REFERENCIAL

1. PLANTEAMIENTO DEL PROBLEMA

1.1. FORMULACIÓN DEL PROBLEMA

Cuando las computadoras evolucionan a máquinas de propósito general mediante sistemas operativos y lenguajes de programación, un gran número de personas de las diferentes ramas aprenden lenguajes de programación y desarrollan aplicaciones según sus necesidades, es decir un esquema donde el desarrollador es también el cliente, por lo tanto si el software desarrollado tenía alguna deficiencia, la misma persona que detecta la falencia es quien lo arregla o elabora las nuevas características de software, por supuesto la gran utilidad de los computadores hace que su uso se extienda y que pronto personas que no entienden nada de lenguajes de programación apoyen su trabajo en software y que las utilidades y funciones de los sistemas crezcan exponencialmente, entonces aparece la necesidad de formalizar los procesos de elaboración de software, naciendo así la ingeniería de software que ha

llegado a especificar principios, métodos y técnicas para el desarrollo de software de calidad, pero para llegar a esta calidad se debe superar uno de los graves problemas que tiene el desarrollo de software que es el poder llegar a un conjunto de especificaciones de software completo y que nos lleve a un alto grado de satisfacción del cliente, y es precisamente la marcada insatisfacción combinada con los altos costos de producir software lo que generó la denominada Crisis del Software (finales de los 60’s e inicio de los 70’s), ante esta necesidad se crearon diferentes técnicas que pretendieron eliminar este serio inconveniente

Las primeras técnicas de especificación aparecen como descripción de procedimientos, a través de seudo códigos como español estructurado y los diagramas de flujo estas técnicas persiguen describir los detalles algorítmicos de un proceso que recorrían los datos para transformase en información, estos dos métodos se utilizaron mucho en la programación estructurada y son muy útiles para describir pequeños módulos de software que realizan una tarea específica, pero cuando se trata de describir el funcionamiento de sistemas más complejos pierden objetividad, otros métodos para recolectar requerimientos de software que son muy validos son las entrevistas y encuestas que permiten tener una visión más clara de lo que los usuarios de un sistema desean del software, el problema es que estos resultados no son fácilmente verificables, es decir como presentarlo a nuestros clientes y que estos nos digan si lo que el desarrollador a captado es en realidad lo que ellos desean,

En cuanto a Programación Orientada a Objetos han existido diferentes formas de obtener los requisitos de software y como expresarlos dependiendo de cada autor, todos estas metodologías se han unificado en una sola que es la más difundida y usada que es la del Lenguaje de Modelado Unificado (UML), que es utilizado por la metodología Proceso Unificado Racional (RUP) que especifica los requerimientos del usuario a través de los casos de uso que son diagramas que pueden ser entendidos de forma más fácil por las personas que no son del área de sistemas, esta metodología es muy útil para proyectos de software de gran tamaño, para proyectos pequeños el esfuerzo en mantener toda la documentación y control que exige RUP llega a ser prohibitivo de usarlo en su totalidad.

En nuestro medio los proyectos de software que se emprenden generalmente son pequeños o medianos, con equipos de desarrollo que generalmente comprenden no más de 3 personas lo que hace muy difícil aplicar eficientemente el modelo de RUP, especialmente en los trabajos de tesis de grado y de prácticas preprofesionales.

Identificándose entonces la necesidad de desarrollar un proceso investigativo que permita crear una metodología que permita realizar especificaciones de software que persigan la calidad del software en proyectos pequeños y medianos.

Las bases para un desarrollo de software exitoso son sin duda alguna los requisitos de software, por más riguroso que sea el proceso de producir los

artefactos software, y por más exitosa que sean las buenas prácticas, estas se quedan sin piso si lo que van a construir no es lo que el usuario del sistema quiere, es decir se llegará a tener un muy buen producto creado de forma muy eficiente pero que hace y tiene lo que nadie quería, por tal motivo la especificación de requerimientos es la base fundamental de todo desarrollo de software.

La concepción general es que los requisitos de software se lo hace al inicio y luego de esto se los sigue rigurosamente, pues estos no deberían variar, esto sería verdad en un ambiente ideal, la verdad es que conforme se avanza el desarrollo del sistemas la mayoría de los requisitos aparecen y se van modificando continuamente, sin duda alguna esto lleva a que el costo del software se incremente y que la calidad del mismo se vea afectada principalmente cuando se tiene un límite de tiempo que esta por expirar, este es el caso real del desarrollo de tesis de grado y proyectos de prácticas preprofesionales.

El desarrollo y formulación de requisitos de software debería responder al gráfico 1.1.

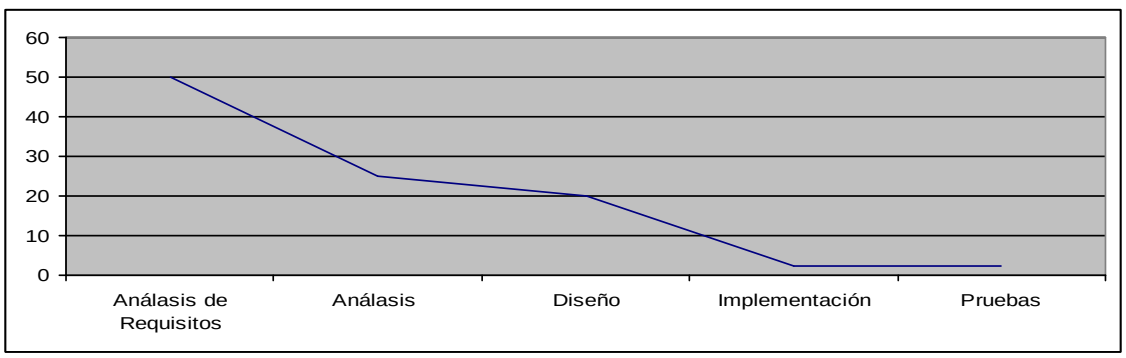


Gráfico 1.1: Desarrollo y Formulación de Requisitos

El esfuerzo en definir los requisitos de software deberán estar distribuidos en toda las etapas de desarrollo del software, pero la mayoría de ellos deben estar claramente definidos en el Análisis de Requisitos y en el Análisis, en el diseño se debería tan solo dar pequeñas modificaciones a los mismos y en el resto de etapas solo adecuaciones pequeñas. Nótese que en la figura 1 aparece el análisis de requisitos como un etapa más del desarrollo de software, esto es debido a su importancia esta etapa estaría dedicada al entendimiento del problema por parte de los desarrolladores, la etapa de análisis sería la formalización de este entendimiento y la presentación a los usuarios para llegar a acuerdos, refinarlos y dejarlos listos para proceder al diseño.

A pesar del estudio de la ingeniería de software y al conocimientos de procedimientos, buenas prácticas, notaciones, etc., no se ha podido dejar de lado la costumbre de “empezar programando”, se ve a la documentación que debe acompañar a un desarrollo de software como un trabajo adicional, muchas veces se realiza el trabajo inverso, es decir se empieza programando y luego dependiendo del resultado se realiza la documentación, esto ha causado una baja calidad de el software y por su puesto las aplicaciones generadas se usan por muy poco tiempo y luego son desechadas.

Ante este esquema de desarrollo de software es imperativo crear una metodología con pasos definidos y con buenas prácticas de software que permita generar requisitos de usuario bien definidos y aceptados por los clientes, los cuales sirvan de bases para el desarrollo de toda la aplicación y que a su vez sea una metodología práctica para la especificación de requisitos

para proyectos medianos y pequeños que permita alcanzar un software que satisfaga las necesidades de los clientes y que sea usado por los mismos para sus labores diarias y no se conviertan en tan sólo un requisito para logar la graduación en la escuela de ingeniería en sistemas de la Universidad Estatal de Bolívar.

1.2 JUSTIFICACIÓN DE LA INVESTIGACIÓN.

La especificación de requisitos de software es una ardua tarea, ya que son la base sobre la cual se construirá toda la aplicación y el nivel en el cual se satisfagan los requisitos definidos determinará al final la satisfacción del cliente.

La especificación de requisitos de software se entendía en un principio como un conjunto de tareas de recolección de información que se realizaban al inicio de la vida del software, es decir hasta que el desarrollador conozca el problema a resolver y después solo restaba analizar, diseñar y luego implementar la aplicación en base a estos conocimientos adquiridos en un principio, es decir considerar a los requisitos de software como un elemento estático durante la vida del desarrollo del software, esta visión es un escenario ideal, que no considera muchos parámetros propios de la realidad de las diferentes organizaciones, la realidad del desarrollo de software es que los requisitos generalmente son cambiantes los cuales deben ser desarrollados en condiciones de tiempo de costo muy restringidos, en este caso el estudio de los requisitos de software debe ser constante y existir durante toda el ciclo de vida del software, a pesar de esta realidad se debe considerar que la mayoría de

requisitos deben ser identificados en las primeras etapas de desarrollo y luego irse puliendo durante el transcurso del desarrollo de la aplicación.

En los trabajos de grado y de prácticas pre profesionales de la carrera de Sistemas de la Universidad Estatal de Bolívar se tiene un número de desarrolladores muy limitado, los cuales desarrollan proyectos de software pequeños y medianos, entonces se hace muy difícil conformar un equipo que pueda realizar las diferentes tareas y que además pueda evaluar las mismas pues terminarían siendo juez y parte, entonces ¿cómo lograr que estos proyectos se desarrollen de tal forma que satisfagan las exigencias de los clientes y que a su vez mantengan un criterio de calidad? La respuesta a esta pregunta es la creación de un metodología que formalice la forma de trabajo empírica que mantienen los alumnos desarrolladores en sus trabajos de grado, aportando a ésta criterios de ingeniería de software y buenas prácticas de desarrollo de software que permitan realizar esta tarea con poco personal, presupuesto y tiempo, evaluando los resultados se sabrá si esto es posible o no.

CAPÍTULO II

MARCO TEÓRICO

2.1 INGENIERÍA DE SOFTWARE.

"Es una disciplina o área de la Informática o ciencias de la computación, que ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelve problemas de todo tipo".1

2.2 SOFTWARE.

Existen muchas definiciones de software, desde la más básica que es el software es la parte intangible de un computador hasta complejas conceptualizaciones que incluyen criterios de calidad, usabilidad y otros, desde el punto de vista de la ingeniería de software el software se puede definir

[1] ROGER PRESSMAN, Ingeniería del Software Un Enfoque Práctico, Editorial Mc Graw Hill. Quinta Edición, 2002 pág. XXIX

como: el producto a obtener, es el resultado de un proceso de ingeniería que permite la creación de un producto que es la salida esperada.

2.2.1 CICLO DE VIDA.

El proceso que se sigue para construir, entregar y hacer evolucionar el software, desde la concepción de una idea hasta la entrega y el retiro del sistema. Tiene que ser Confiable, predecible y eficiente, existen diferentes ciclos de vida en dependencia del enfoque que se le ha dado al desarrollo del software, en todos estos se puede ver el análisis de requerimientos como una de las etapas de los ciclos.

2.3. REQUERIMIENTO

"Un requerimiento es una característica del sistema o una descripción de algo que el sistema es capaz de hacer con el objeto de satisfacer el propósito del sistema"2.

Un requerimiento es una transacción visible por un usuario, es decir un comportamiento que el sistema tiene ante un estímulo específico que una entidad externa activa, estas unidades externas pueden ser: usuarios, otros sistemas, diversos eventos, etc.

[2] LAWRENCE PFLEEGER SHARI, Ingeniería de Software Teoría y Práctica, Editorial Prentice Hall, Primera Edición 2002, página 156

Los requerimientos pueden dividirse en Funcionales y No Funcionales, todos los requerimientos deben ser extraídos del cliente de forma sistematizada y documentada.

2.3.1. REQUERIMIENTOS FUNCIONALES

Son los requerimientos que son percibidos por los usuarios, es decir las funcionalidades que un sistema tiene que ejecutar, estos requerimientos no dependen de restricciones.

2.3.2. REQUERIMIENTOS NO FUNCIONALES

Estos requerimientos son dados por restricciones que son impuestas al sistema, las restricciones puede ser de diferentes tipos (tiempo, hardware, tipos de archivos, etc.)

Los requerimientos no funcionales limitan la solución que el equipo de desarrollo puede dar a un problema, por ejemplo si estamos obligados a utilizar hardware específico, o si tenemos que generar archivos de texto plano, etc.

2.4. METODOLOGÍAS PARA LA ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE

Debido a la importancia de la especificación de requisitos se han definido diferentes técnicas para generar un conjunto de requisitos estables y aceptados que permitan desarrollar software que sea aceptado por los usuarios finales.

2.4.1. METODOLOGÍA DE LA INGENIERÍA DE REQUERIMIENTOS DE SOFTWARE (SREM, SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY)

Esta metodología se desarrollo para generar requisitos de software de tiempo real, en estos sistemas los requisitos no funcionales son primordiales y es de suma importancia describirlos de forma clara.

Esta metodología se define en dos etapas que son:

1. Especificación de requisitos.

   En esta fase se empieza escribiendo los requisitos de forma gráfica, generando los diagramas de red de requerimientos (gráfico 2.1.), luego se utiliza el Lenguaje de enunciados de requerimientos (Requirements Engineering Language), para traducir la red en un RSL, que es un seudo código como podemos ver en el gráfico 2.2.

2. Análisis y generación de reportes.

   Los RSL generados en la fase anterior son analizados por un Sistema de Validación de Requerimientos (REVS, Requirements Engineering Validation System), mediante el cual genera diferentes reportes.

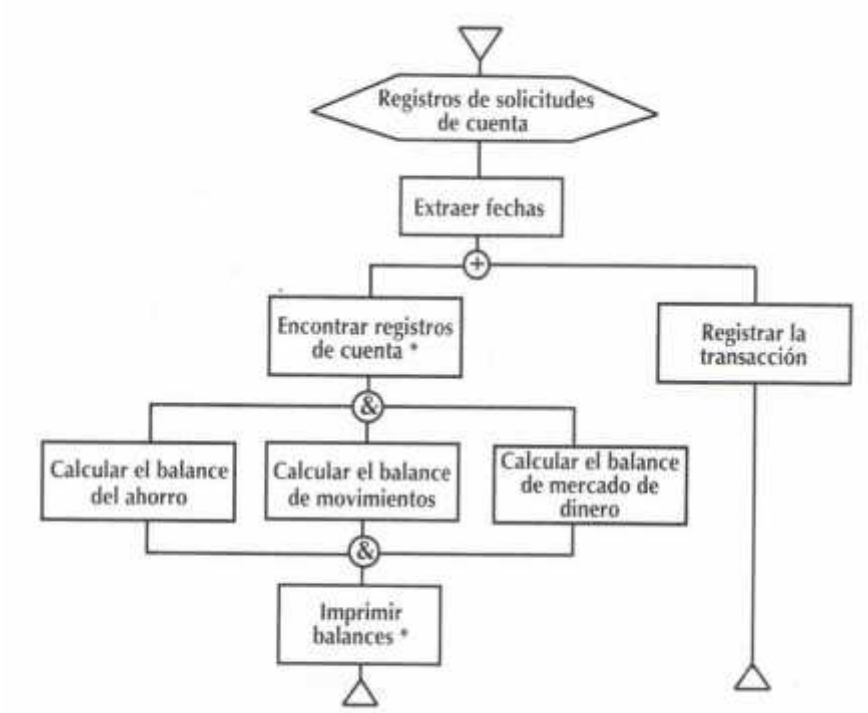


Gráfico 2.1: Red de Requerimientos

FUENTE: LAWRENCE PFLEEGER SHARI, Ingeniería de Software Teoría y Práctica, Editorial Prentice Hall, Primera Edición 2002, página 187

```
R_NET: PROCESS_TRANSACTION
 STRUCTURE:
   INPUT_INTERFACE_ACCOUNT_REQUEST_RECORD
   EXTRACT_DATES
   DO (REQUEST=TRANSACTION)
    RECORD_TRANSACTION
    TERMINATE
   OTHERWISE
    FIND_ACCOUNT_RECORDS
    COMPUTE_SAVINGS_BALANCE
    AND COMPUTE_CHECKING_BALANCE
    AND COMPUTE_MONEY_MARKET_BALANCE
    PRINT_BALANCES
    TERMINATE
   END
END
```

Gráfico 2.2: Seudo Código SRL

FUENTE: LAWRENCE PFLEEGER SHARI, Ingeniería de Software Teoría y Práctica, Editorial Prentice Hall, Primera Edición 2002, página 188

2.4.2. EIA / IS – 632.

Este es un estándar desarrollado como un proyecto adjunto de Electronic Industries Alliance (EIA) e internacional Council on Systems Engineering (INCOSE), define una aproximación sistemática a la ingeniería y reingeniería de un sistema, incorporando mejores prácticas.

Este estándar es aplicable para todo el ciclo de vida de software y por supuesto define como se deben capturar los requisitos de software de la siguiente forma:

En el proceso de diseño del sistema se utilizan dos sub procesos que permiten transformar los requerimientos adquiridos en un conjunto de productos realizable, estos son: Definición de Requerimientos y Definición de la Solución la relación es estos sub procesos se definen en el gráfico 2.3.

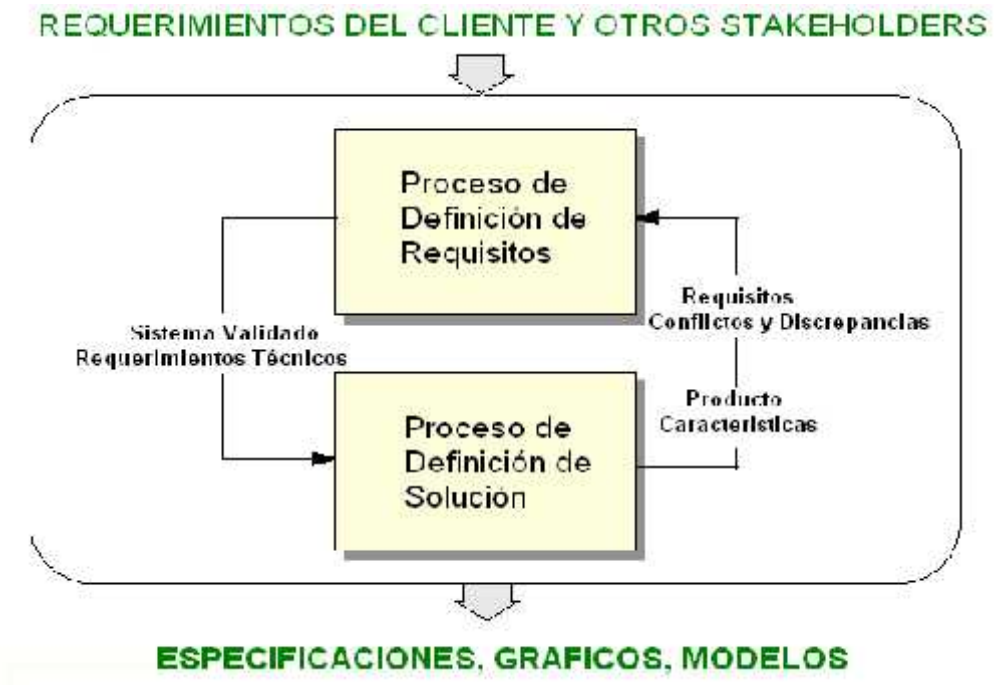


Gráfico 2.3: Proceso del Diseño del Sistema

FUENTE: ANSI/EIA-632-1998 Approved: January 7, 1999

2.4.2.1. PROCESO DE DEFINICIÓN DE REQUISITOS

Las entradas a este proceso son de tres tipos:

a. Requerimientos desde: la aceptación, otros documentos, individuos o grupos que tienen interés asegurar el resultado de la ingeniería o reingeniería del sistema.

b. Requisitos en forma de resultados de otros procesos como: planes técnicos y decisiones desde las revisiones técnicas.

c. Peticiones o aprobaciones de cambios en los requerimientos del primer tipo.

El desarrollador debería planificar y apropiarse de tareas que completen los requisitos, estas tareas son:

a. Identificar, recoger y asignar prioridades, clientes, usuarios u operadores de requisitos para el sistema o una porción de ellos incluyendo algunos requisitos para desarrollo, producción, test, despliegue e instalación, entrenamiento, operaciones, soporte y mantenimiento y disponibilidad de productos del sistema.

b. Asegurar que el conjunto de requisitos aceptados son lo que el cliente necesita y espera.

2.4.2.2. PROCESO DE DEFINICIÓN DE LA SOLUCIÓN

Este proceso es usado para generar un diseño aceptable de la solución, la que debe satisfacer:

a. Requerimientos técnicos del sistema resultado de un completo proceso de definición de requisitos descrito anteriormente.

b. Los requerimientos técnicos obtenidos en el proceso de definición de la solución descrita en la iteración anterior.

El desarrollador debería planificar y apropiarse de tareas para completar los requisitos tales como:

a. Seleccionar e implementar una o más aproximaciones proveyendo de una solución abstracta de la solución de los requisitos técnicos del sistema.

b. Estableciendo un conjunto de soluciones lógicas para: analizar; identificar y definir interfaces; estados y modelos; líneas de tiempo; datos; flujos de control; analizar comportamientos y analizando fallas y definiendo efectos de estas.

c. Asignar requerimientos técnicos a elementos de la representación de la solución lógica.

d. Identificar y definir requerimientos técnicos generados en las tareas a y b, asegurando que estos sean aceptados.

e. Almacenar los resultados del conjunto de soluciones lógicas, normalmente estos se almacena en una base de datos del proyecto.

El desarrollador debe generar alternativas de solución para:

1. Identificación y definición de interfaces físicas.
2. Identificación y análisis de parámetros críticos
3. Identificación y evaluación de opciones de soluciones físicas.
4. Rendimiento del análisis del sistema.

La información presentada en el subtema 2.4.2 EIA / IS – 632 ha sido tomada de ANSI/EIA-632-1998 Approved: January 7, 1999, pero no como una copia literal.

2.4.3. IEEE STD 1233, EDICIÓN 1998. GUÍA PARA EL DESARROLLO DE ESPECIFICACIONES DE REQUERIMIENTOS DE SISTEMAS

"Esta guía nos de la pauta para el desarrollo de un conjunto de requerimientos que satisfarán una necesidad específica. En esta guía, a ese conjunto de requerimientos se le denomina Especificación de Requerimientos del Sistema (System Requirements Specification, SyRS). El desarrollo de una SyRS incluye la identificación, organización, presentación y modificación de los requerimientos. Esta guía trata las condiciones necesarias para incorporar conceptos operacionales, restricciones de diseño, y requerimientos de la configuración del diseño en la especificación. Además, trata las características y cualidades necesarias de los requerimientos individuales y del conjunto de todos los requerimientos."[3]

[3] **IEEE Std 1233, Edición 1998, página 1**

2.4.3.1. ESPECIFICACIÓN DE REQUERIMIENTOS DEL SISTEMA

"Una SyRS tradicionalmente ha sido vista como un documento que comunica los requerimientos del cliente a la comunidad técnica que especificarán y construirán el sistema. La colección de requerimientos que constituyen la especificación y su representación actúan como el puente entre los dos grupos y debe ser entendible tanto por el cliente como por la comunidad técnica. Una de las tareas más difíciles en la creación de un sistema, es aquella de comunicar a todos los subgrupos, especialmente en un solo documento. Este tipo de comunicación usualmente requiere diferentes formalismos y lenguajes."[4]

Este estándar sugiere los siguientes ítems:

a. Definición

   Esta es la descripción de los que los clientes esperan, debe describirse el perfil del uso del sistema y los parámetros necesarios.

b. Propiedades

   Los requerimientos deben cumplir con las siguientes propiedades:

   a) "Conjunto Único. Cada requerimiento debe ser declarado sólo una vez.

   b) Normalizado. Los requerimientos no se deben traslapar (Por ejemplo, no se deben referir a otros requerimientos ni a las capacidades de otros requerimientos).

[4] **IEEE Std 1233, Edición 1998, página 4**

c) Conjunto interdependiente. Se deben definir explícitamente las relaciones entre los requerimientos individuales para mostrar cómo los requerimientos están relacionados para formar el sistema completo.

d) Completo. Una SyRS debe incluir todos los requerimientos dados por el cliente, así como aquellos requerimientos necesarios para la definición del sistema.

e) Consistente. El contenido de una SyRS debe ser consistente y sin contradicciones en el nivel de detalle, estilo de la declaración de los requerimientos y en la presentación del material.

f) Acotado. Los límites, alcance y el contexto de los requerimientos del sistema deben ser identificados.

g) Modificable. La SyRS debería ser modificable. Requerimientos claros y sin traslapamientos contribuyen a lograrlo.

h) Configurable. Las versiones deberían ser mantenidas a través del tiempo y a través de las instancias de la SyRS.

i) Granular. Éste debe ser el nivel de abstracción para el sistema que está siendo definido"[5]

c. Propósito

Provee de una descripción del sistema en términos de iteraciones, debe describir entradas y salidas-

[5] **IEEE Std 1233, Edición 1998, página 5**

“Esta cláusula proporciona una descripción de los pasos en el proceso de desarrollo de la SyRS. El proceso de desarrollo de los requerimientos del sistema, en general, se relaciona con 3 agentes externos (cliente, ambiente y comunidad técnica) Cada uno de estos agentes es descrito en la figura siguiente:” [6]

Gráfico 2.4: Contexto de Desarrollo de los SRS

FUENTE: IEEE Std 1233, Edición 1998

Este estándar es se desarrollo para especificación de requisitos para mayor referencia se puede leer el ANEXO 2 IEEE Std 1233, Edición 1998.

2.4.4. Modelo de Capacidad y Madurez o CMM (*Capability Maturity Model*)

CMM es un modelo de evaluación de los procesos de una organización que se desarrollo inicialmente para los procesos relativos al desarrollo e implementación de software por la Universidad Carnegie-Mellon para el SEI (Software Engineering Institute).

[6] **IEEE Std 1233, Edición 1998, página 8**

Este modelo se fundamenta en establecer las KPA (key Process Area), en cada area se define un conjunto de buenas prácticas las cuales deben: ser definidas documentadamente, deben tener los recursos necesarios (materiales y de capacitación), ejecutadas de forma sistémica, universal y uniforme, medidas y verificadas, en cada área se debe alcanzar un nivel de madurez, el cual puede variar en 5 distintos niveles que son:

1. Inicial

   Las organizaciones no mantienen un ambiente estable para el desarrollo y mantenimiento del software, los procesos son llevados de forma empirica y sin planificación.

2. Repetible

   Las organizaciones tienen prácticas definidas, en las diferentes áreas se documenta, se evalúa y se hace un seguimiento limitado de la calidad, pero adolece de comunicación entre las áreas.

3. Definido

   Se mantiene una buena gestión en las áreas y además se mantiene una coordinación entre las mismas, se puede hacer evaluaciones y métricas más exactas y las áreas pueden coevaluarse.

4. Gestionado

   Se mantienen el uso de métricas para calidad y productividad, las cuales se usan de modo sistematizado, permitiendo tomar decisiones adecuadas para mitigar riesgos, obteniendo software de alta calidad.

5. Optimizado

   Se mantiene un buen nivel de gestión pero además se retroalimenta los resultados de las métrica permitiendo que toda la organización se centre en la mejora continua, generando software de altísima calidad.

Este modelo es ideal para empresas de desarrollo de software que están generando contantemente nuevas versiones de un sistema o diferentes sistemas de forma continua, el esfuerzo de implantar CMM es muy alto en costo y tiempo, por lo cual sólo se justifica si la empresa desarrolladora se mantiene en el mercado por un largo tiempo.

2.4.5. Proceso Unificado Racional (Rational Unified Process RUP).

Este es un Proceso de Ingeniería de Software que provee de una forma disciplinada de asignar tareas y responsabilidades entre los miembros del equipo de desarrollo, el objetivo de este proceso es asegurar la producción de software de alta calidad, con conocimiento de las necesidades de los usuarios finales, con cronograma y presupuesto predecibles, este proceso es desarrollado y mantenido por Rational Software.

RUP trata de mejorar la productividad del equipo, proveyendo a cada miembro de un fácil acceso a la base del conocimiento, con directrices, plantillas y herramientas mentoras para todas las actividades criticas del desarrollo, de tal forma que todo el equipo accede a la misma base del conocimiento sin importar si trabajan en requerimientos, diseño, test, administración del proyecto o

administración de la configuración, asegura que todos los miembros del equipo comparten lenguajes comunes, procesos y vistas de cómo se desarrolla el software. Las actividades del RUP se enfocan en crear y mantener modelos que se centran en la producción evitando la acumulación de papeles, para esto utiliza el UML.

UML (Unified Modeling Language) es Lenguaje de Modelado Unificado, que fue desarrollado por Rational Software y que luego se adoptó como un estándar de la industria, su objetivo principal es comunicar de forma clara requerimientos, arquitecturas y diseños.

El RUP es soportado por herramientas, las cuales asisten el desarrollo de software, que se usan para crear y mantener los artefactos visuales del proceso de ingeniería de software.

El RUP es un proceso configurable tanto para pequeños equipos de desarrollo así como para equipos muy grandes de desarrollo, a través de un simple y claro proceso arquitectónico, este proceso se puede adaptar a diferentes situaciones, logra esto aplicando un conjunto de buenas prácticas.

2.4.5.1 Buenas Prácticas para el desarrollo efectivo con RUP

El RUP describe las siguientes buenas prácticas para los equipos de desarrollo de software:

- **Desarrollo de Software Iterativo**

  Es desarrollo de software no puede ser secuencial, es decir primero definimos el problema entero, diseñamos la solución, construimos el software y realizamos las pruebas y el producto termina, esta visión lineal no es real, el desarrollo de software es un conjunto de iteraciones que permiten incrementar el entendimiento del problema a través de sucesivos refinamientos, generando una solución efectiva mediante múltiples iteraciones, de cada una de estas obtenemos una versión ejecutable, lo que permite que el equipo de desarrollo se enfoque en producir resultados.

- **Manejo de Requerimientos**

  El RUP define como: describir, organizar y documentar requisitos funcionales y restricciones; registra y almacena documentos de reglas del negocio y decisiones; y; captura y comunica los requerimientos del negocio. Las nociones de Casos de Uso y Escenarios proveen al proceso de una excelente forma de capturar requerimientos funcionales para asegurar la conducción del diseño, implementación y pruebas, haciendo que el producto terminado sea el más esperado por los usuarios finales, provee de un vía coherente y traceable a través del desarrollo y liberación del sistema.

- **Uso de Componentes Basados en Arquitecturas**

  EL RUP se enfoca en un desarrollo claro y establece líneas bases para una robusta arquitectura ejecutable, priorizando el comprometimiento de recursos para todo el desarrollo. Describe como diseñar una fuerte

arquitectura que sea flexible, adaptable a los cambios y entendible, prometiendo una fácil reutilización del software, es decir un desarrollo de software basado en componentes no triviales, subsistemas que realizan funciones limpias, pudiendo utilizar componentes nuevos y existentes que son ensamblados con arquitecturas bien definidas que permiten generar aplicaciones a la medida.

- **Modelos de Software Visuales**

  Se basa en el mantenimiento y creación de modelos visuales que presentan como capturar la estructura y el comportamiento de arquitecturas y componentes, esto permite ocultar detalles y escribir el código usando bloques de construcción gráficos. Las abstracciones ayudan a comunicar diferentes aspectos del software, ver como los elementos del sistema actúan juntos, haciendo que la construcción con bloques sea consistente con el código, dando equilibrio entre el diseño y la implementación promoviendo comunicaciones no ambiguas, todo esto lo realiza mediante el UML

- **Calidad del Software Verificable**

  Una pobre fiabilidad y un pobre rendimiento de los sistemas, son factores comunes que impiden la aceptación de aplicaciones de software, La calidad podría ser revisada con respecto a los requerimientos basados en la fiabilidad, funcionalidad, rendimiento de la aplicación y rendimiento del sistema, el RUP asiste en la planificación, diseño, implementación, ejecución y evaluación de las pruebas. La calidad se construye en el proceso, en todas las actividades que involucran a todos los participantes, usando objetivos medibles a

criterios, que no tratan a las actividades de rendimiento como un grupo separado.

- **Control de cambios de Software**

  La habilidad de manejar cambios es aceptar que un cambio es posible y se puede seguir, pues los cambios en un ambiente de desarrollo son inevitables. El RUP describe como controlar, grabar y monitorear los cambios para poder llegar a un desarrollo iterativo exitoso. También guía al equipo de desarrollo a mantener espacios de trabajo estables y seguros para cada desarrollador proveyendo aislamiento de los cambios realizados en otro espacio de trabajo y controlando los cambios de todos los artefactos de software. Describe como llevar al equipo trabajando como una unidad refiriendo como automatizar la integración y administrando la construcción del software.

2.4.5.2. Perspectiva del Proceso

El proceso puede ser descrito en dos dimensiones y dos ejes:
El eje horizontal representa el tiempo y presenta el aspecto dinámico del proceso, es decir cómo actúa y esta expresado en términos de ciclos, fases, interacciones e hitos

El eje vertical representa el aspecto estático del proceso, es decir como el proceso se describe en términos de actividades, artefactos, trabajadores y flujos de trabajo.

EL EJE HORIZONTAL

Este eje es la organización dinámica del proceso a lo largo del tiempo, el ciclo de vida es dividido en fases, cada fase trabaja en una nueva generación del producto. El RUP divide el ciclo de desarrollo en cuatro fases consecutivas que son (ver gráfico 2.5.):

1. Fase de Inserción
2. Fase de elaboración
3. Fase de construcción
4. Fase de transición

EL EJE VERTICAL

Este eje describe la vista estática del proceso describe quien está haciendo qué, cómo y cuándo, el RUP representa elementos del modelos que son Trabajadores, Actividades, Artefactos y Flujos de trabajo (ver gráfico 2.5.)

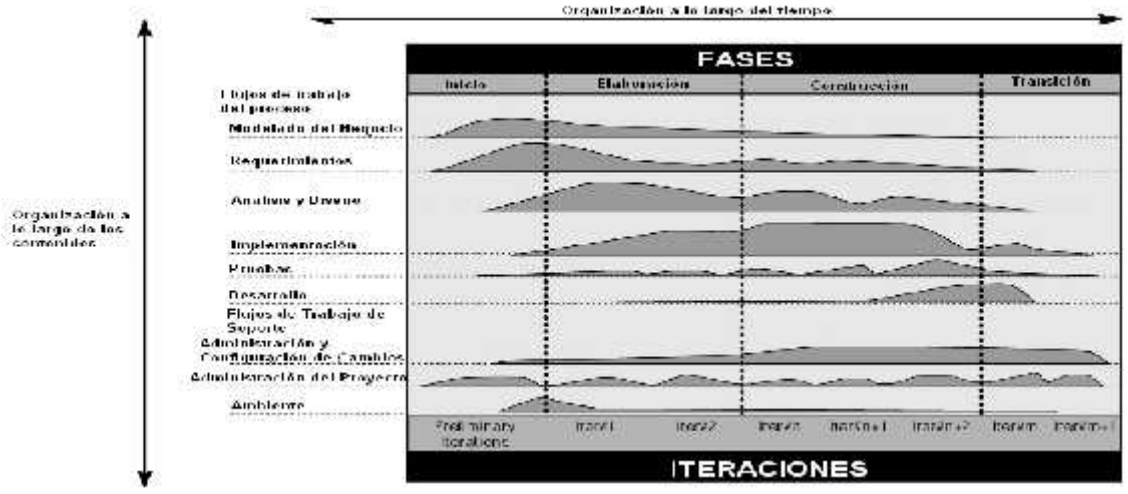


Gráfico 2.5.: Grafico del Modelo Iterativo, presenta el proceso estructurado en dos ejes

FUENTE: Rational Software White Paper TP026B, Rev 11/01

2.4.5.2. ESPECIFICACIÓN DE REQUISITOS EN RUP

El RUP a través del UML permite representar la funcionalidad del sistema mediante los casos de uso, esta técnica permite capturar los requisitos en función de la importancia que estos tiene para el usuario.

Los casos de uso además de permitir la especificación gráfica de los requisitos del sistema también guían al desarrollador durante todo el proceso como se puede ver en el gráfico 2.6.

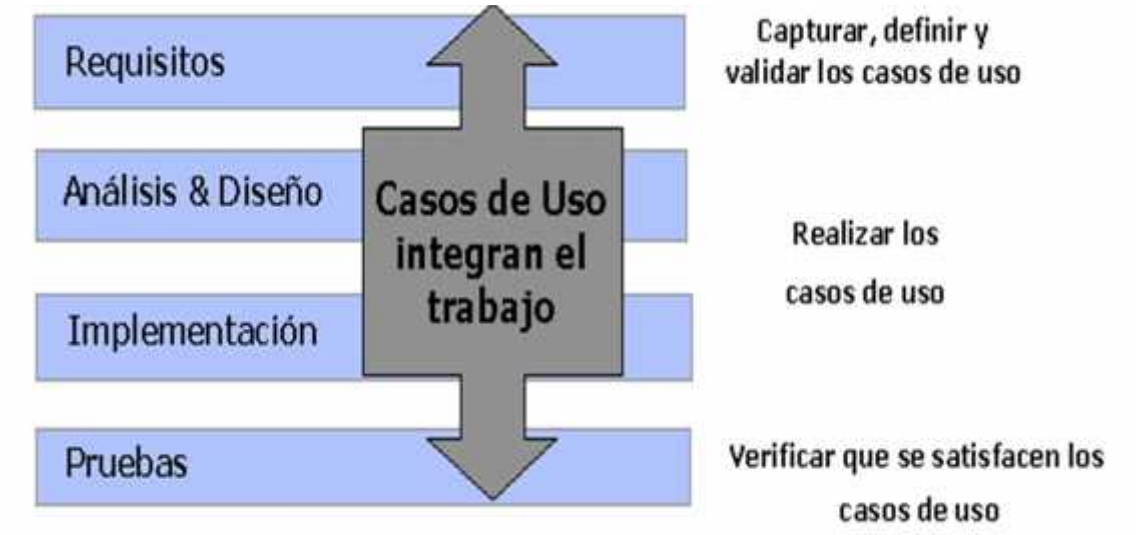


Gráfico 2.6: Casos de Uso a Través de RUP

FUENTE: DESARROLLO DE SOFTWARE UTILIZANDO LA METODOLOGÍA RUP (Process Unified Rational)

Caso Práctico: “Sistema Escolástico Parametrizable

Para mejorar la especificación de requisitos generados a través de los casos de uso se puede definir un estándar de documentación, el cual permite documentar y hacer el seguimiento de las versiones de Especificaciones de Requisitos de Software, este documento se puede ver en el anexo 2

Para mayor referencia se puede referir al anexo 4 **Rational Software White Paper TP026B, Rev 11/01**

2.5. RESUMEN DE LAS METODOLOGÍAS DE ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE Y SU APLICABILIDAD

En la tabla 2.1. se resumen las diferentes características de las metodologías estudiadas y su aplicabilidad en los sistemas desarrollados en las tesis y pasantías en la Escuela de Sistemas de la Universidad Estatal de Bolívar.

CARACTERÍSTICA	METODOLOGÍAS				
	SREM	EIA / IS 632	IEEE STD 1233	RUP	CMM
Tipo de sistema	Tiempo Real Cualquiera	Cualquiera	Cualquiera	Cualquiera	Cualquiera
Tamaño de la organización	Mediano	Adaptable	Adaptable	Adaptable	Grande
Tiempo de Implantación	Mediano	Mediano	Mediano	Mediano	Alto
Lenguaje de Especificación	Diagramas de red y RSL	Adaptable	Adaptable	UML	Adaptable
Orientación	Estructurada	No especificada	No especificada	Objetos	Procesos
Nivel de Familiarización	Ninguno	Bajo	Bajo	Medio	Ninguno

Tabla 2.1.Resumen de las Características de las Metodologías Estudiadas

Metodología de la Ingeniería de Requerimientos.de Software (SREM) es una metodología creada para sistemas en tiempo real donde las restricciones son muy altas y requieren ser documentadas y rastreadas, sin embargo se puede utilizar para sistemas de propósito general.

Los estudiantes no están familiarizados con esta metodología, además esta fue creada para proyectos de orientación estructurada, por lo tanto no es aplicable a los sistemas desarrollados en la Universidad estatal de Bolívar.

Los estándares EIA / IS 632 y IEEE STD 1233 aunque se los trata en las asignaturas de Ingeniería de Software mantienen un nivel de familiarización bajo, además no poseen lenguaje de especificación definido, por lo tanto es difícil su aplicación.

RUP este proceso es más conocido por los estudiantes, tiene un lenguaje de especificación definido es adaptable al tamaño de la organización, sin embargo para cubrir los diferentes roles que define RUP se necesita de al menos diez personas para una mejor aplicación del mismo, esto hace que no sea aplicable en los proyectos desarrollados.

CMM es una metodología que trata de optimizar los procesos con el objetivo de que una organización alcance niveles superiores de madurez, pero para que una organización madure se necesita un alto tiempo de implantación, por esto no es aceptable para desarrollos de software que en promedio tienen u8na duración de seis meses.

En base a lo expuesto anteriormente se hace necesario establecer una metodología que los alumnos puedan aplicar en los desarrollos de software medianos y pequeños que se ejecutan en la Escuela de Sistemas de la Universidad Estatal de Bolívar.

CAPÍTULO III

MARCO PROPOSITIVO

3.1. PROPUESTA METODOLÓGICA PARA LA ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE PARA PROYECTOS PEQUEÑOS Y MEDIANOS

La mayoría de los proyectos de software pequeños y medianos se basan en mantener una base de datos es decir ingresos, borrados, actualizaciones en las diferentes tablas de una base de datos y el cuidado de la integridad de la misma, con estas consideraciones podemos establecer los siguientes pasos para la especificación de requisitos.

3.1.1. Definición de pasos a seguir

3.1.1.1. Entendiendo el problema

Este es un principio practicado en cualquier solución de problemas, es básico entender cuál es el problema y en el caso de proyectos medianos y pequeños de software se debe establecer: cuál es la situación actual, delimitar lo que se va a automatizar desde

una visión global viendo al problema como un sólo elemento donde se pueda establecer específicamente entradas y salidas, es decir que datos necesita la aplicación y quién los proporciona; qué información genera y para quién. Lo anteriormente mencionado puede ser representado en un diagrama de casos de uso general.

3.1.1.1.1. Construyendo el Diagrama de Casos de Uso general

El primer Diagrama de Casos de Uso general debe ser construido de la siguiente forma:

a. Observación.- Es un método imprescindible que permite observar el sistema actual (manual o a través de software), se trata de determinar qué datos se ingresan al sistema y qué productos se generan, en este momento no es importante como se procesa la información.

b. Encuestas.- esta técnica de recolección de datos es válida si solo si la población de futuros usuarios del sistema es numerosa, generalmente en proyectos pequeños o medianos las poblaciones numerosas tienen acceso a una serie de reportes y rara vez realizan ingresos o modificaciones.

c. Entrevistas.- esta herramienta es vital especialmente para los usuarios que tendrán la responsabilidad de ingresar los datos, quienes generalmente también consumen la información generada.

Todas las herramientas listadas anteriormente deben estar orientadas solo a determinar cuáles son los datos ingresados al futuro sistema y los productos que cada usuario espera obtener del sistema, así como las características que deberán tener los productos, en este momento es importante tener claro que los detalles de cómo se procesa la información no son el objetivo.

Con la información recolectada se podrá construir un primer diagrama de Casos de Uso general, este diagrama deberá ser presentado y explicado a los usuarios, para cada uno de ellos pueda ver los resultados esperados, junto a este diagrama se deberá presentar formatos de ingresos de datos y salida de información, estos formatos reflejarán los datos ingresados con sus característica y los productos de salida con sus respectivas características, este sería un primer modelo que permitirá que los futuros usuarios del sistema tengan una visión clara que será lo que vamos a construir y por supuesto hagan sus observaciones para refinar el modelo.

El diagrama de Casos de Uso general junto con los formatos de entrada y salida una vez depurados por el desarrollador y los futuros usuarios son una representación del sistema que nos permitirá establecer los siguientes pasos.

3.1.1.2. ¿Qué datos se almacenan?

El responder a esta pregunta es muy importante, se debe realizar el diseño de la base de datos, para realizar este trabajo se debe:

Recolectar copias de los documentos almacenados, se debe recolectar todo documento que sea almacenado en el sistema actual, en caso de ser software se debe obtener una copia de la base de datos para estudiarla y construir el diseño de la base de datos en caso de que no exista un manual técnico donde este diseño esté explicado explícitamente.

Revisar las leyes que regulen esta información, esto es algo que generalmente no se lo realiza pero es de vital importancia revisar las leyes y reglamentos que regulen la tenencia o las características de la información a almacenar (ej. facturas con las regulaciones del SRI, notas con las regulaciones del reglamento o estatuto de cada institución).

En base a la información recolectada anteriormente podemos llegar a un primera descripción de la base de datos en la cual ya podemos ver las relaciones existentes entre los datos, esta descripción será expresada buscando enumerar los datos que serán almacenados, en otras palabra estamos hablando de un diseño de base de datos entidad – relación.

A pesar de que la construcción del sistema es orientado a objetos la base de datos será sin duda un construcción entidad – relación, el tratar de responder que datos se almacenan descubrirá también un conjunto de documentos con información que muchos futuros usuarios pudieron omitir por lo tanto se podría considerar construir nuevamente el diagrama de Casos de Uso general (paso 1).

3.1.1.3. Determinación de las funcionalidades del Producto.

Basándose en la información recolectada se debe determinar cuáles son las funciones que debe tener el producto de software, ésta es una primera visión general que deberá ser revisada con los usuarios, cada funcionalidad deberá ser presentada a los posibles usuarios, los gráficos ideales para lograr la comprensión de estas funcionalidades son los Diagramas de casos de uso.

"La vista de caso de uso captura el comportamiento de un sistema, de un subsistema o de una clase, tal como se muestra a un usuario exterior. Reparte la funcionalidad del sistema en transacciones significativas para los actores – usuarios ideales de un sistema. Las piezas de funcionalidad interactiva se llaman casos de uso. Un caso de uso describe una interacción con los actores como secuencia de mensajes entre el sistema y uno a más actores"7

El o los diagramas de caso de uso serán presentados a los usuarios para que estos puedan ver las funcionalidades que han sido entendidas por el equipo de desarrolladores,

3.1.1.4. Caracterización de usuarios

En este momento se tendrá una visión más clara de las funciones que deberá tener el software a desarrollarse, entonces podemos definir quienes son los usuarios finales, a cada grupo de usuarios se le debe presentar el modelo creado en el paso anterior pero

[7] RUMBAUGH , James; JACOBSON Ivar; BOOCH, Grady El Lenguaje Unificado de Modelado. Manual de Referencia, Editorial Pearson Addison Wesley. Segunda Edición, 2004 pag. 54

modificado un poco, cada usuario deberá ver sólo el modelo con las funcionalidades en las cuales él está involucrado, para que este trabajo sea más productivo debemos agrupar a los usuarios por las funcionalidades que utilicen del sistema.

Por ejemplo tomemos un software para notas, en el cual la secretaria es la que ingresa las notas en la base de datos, los alumnos pueden consultar sus notas, el decano y el vicedecano pueden ver las notas y pedir reportes por carrera, curso o alumno, los docentes pueden ver las notas y pedir reportes de notas de cualquier alumno, curso o carrera.

El contexto explicado en el párrafo anterior es muy sencillo pero servirá para caracterizar usuarios. A simple vista se puede distinguir los siguientes usuarios Alumnos, Secretaria, Docentes, Decano y Vicedecano es decir 4 tipos de usuarios, pero no olvidemos que la descripción que queremos hacer es de funcionalidad del sistema, entonces nuestra caracterización será Secretaria (ingresos al sistema), alumnos (consultar sus notas) y decano, vicedecano y docentes se agruparían en una solo tipo de usuario pues todos utilizan las mismas funcionalidades.

Cada grupo de usuarios debe ver el diagrama de casos de uso orientado a sus funcionalidades y aportar con el mismo, aumentando, retirando o especificando de mejor manera los posibles casos de uso, luego de este aporte se beberá presentar un diagrama para que vea el todo el sistema en su conjunto, por supuesto que para usuarios como los alumnos se deberá tomar una muestra representativa, pero si la funcionalidad para estos es muy limitada como la que se explica en el ejemplo podemos omitir este paso para los tipos de usuarios de limitada funcionalidad, otra

forma de tomar esta información es posterior a la primera versión ejecutable del sistema, se puede tomar estos datos a través de un sistema de encuestas, que son útiles para conocer la opinión de un grupo numeroso de usuarios, como por ejemplo en una página Web, por supuesto se supone que cualquiera que sea la opinión de este grupo de usuarios no representará cambios extremos en la aplicación, es decir serán solo pequeños cambios de forma y no de fondo.

Una vez ejecutado estos cuatro primeros pasos tendremos un conocimiento del problema y de los que los usuarios desean del sistema a construirse, además ya se tendrá un diseño de base de datos que permita empezar con la construcción de los objetos para esto debemos realizar los siguientes pasos.

3.1.1.5. Determinación de funcionalidades

Se establecerá que objetos requieren tener funcionalidades básicas. Se entiende como funcionalidades básicas a Inserción, Actualización, Borrado, Búsquedas y Cuidado de Integridad Referencial. La implementación de estas funcionalidades básicas cubrirá muchos de los informes o reportes que requieren los usuarios.

Algunos objetos o reportes requerirán de funcionalidades extras como controles por fechas, aplicación de algoritmos para cálculos y otros, para esto realizaremos:

Casos de Uso.

Los casos de uso ya han sido desarrollados en los pasos anteriores por tal motivo en este momento serán una depuración de los mismos, lo que permitirá establecer junto con los futuros usuarios su completitud y validez.

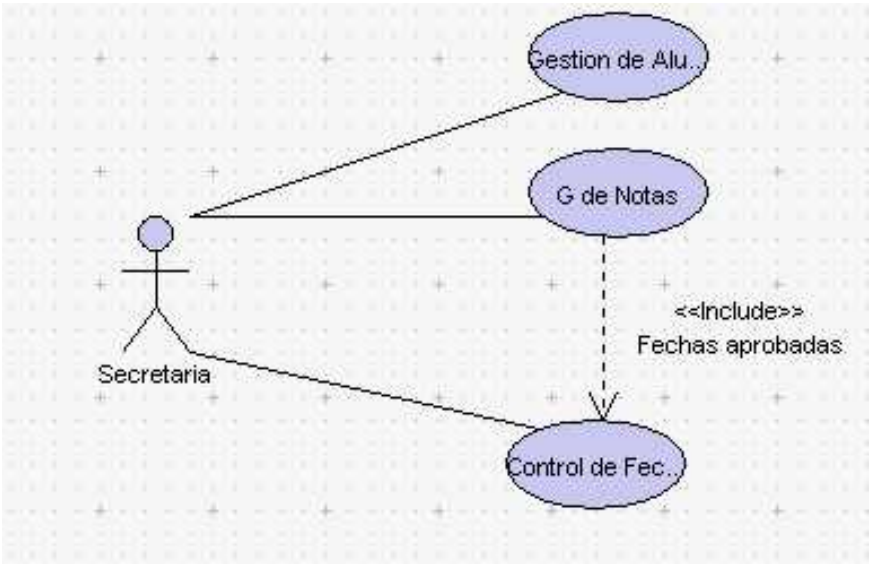


Grafico 3.1. Casos de Uso

Cada Caso de Uso será especificado para determinar si necesita alguna funcionalidad extra por ejemplo:

Gestión de Alumnos

Inserción, Actualización, Borrado, Búsquedas y Cuidado de Integridad Referencial gestionadas por el actor secretaria.

En este caso de uso se observa que no tenemos funcionalidades extra por tal motivo no se necesita mayor especificación pues para estas tenemos parámetros básicos que serán aplicables cualquier caso de uso como son:

- En Inserción y actualización, validación de datos (tipos, formatos, existencias como por ejemplo la cédula no debe repetirse, dependencias funcionales)
- En borrado, validación de datos llave por el cual se buscar el elemento a borrar, dependencias funcionales (borrados en cascada)
- En búsquedas, criterios de búsqueda, por claves, por datos parciales (like) por referencias de otros objetos

Tomemos un caso de uso con funcionalidades extras.

Gestión de Notas

Como se puede ver en le gráfico la gestión de notas incluye el control de fechas, es decir si quiero ingresar notas antes debe existir un calendario establecido que permita o no ingresar las mismas con diferentes autorizaciones por lo tanto:

Inserción, Actualización, Borrado, Búsquedas y Cuidado de Integridad Referencial gestionadas por el actor secretaria.

Funcionalidad extra Calendarización

Para esta funcionalidad debemos especificar el siguiente:

Flujo de eventos para Calendarización

Flujo Básico

Es el flujo de eventos que ocurrirá si todos los pasos se ejecutan sin generar ningún error por ejemplo:

1. Ingresar a la interfaz Ingresar Notas
2. Verificar si la fecha actual está dentro de la Agenda establecida
3. Respuesta Positiva de la Agenda
4. Ingresar las Notas de las asignaturas escogidas
5. Enviar las notas ingresadas
6. Confirmación de inserción exitosa
7. El sistema confirma la inserción en la base de datos

Los pasos descritos se pueden representar en un diagrama se secuencia como muestra la figura 3.2.

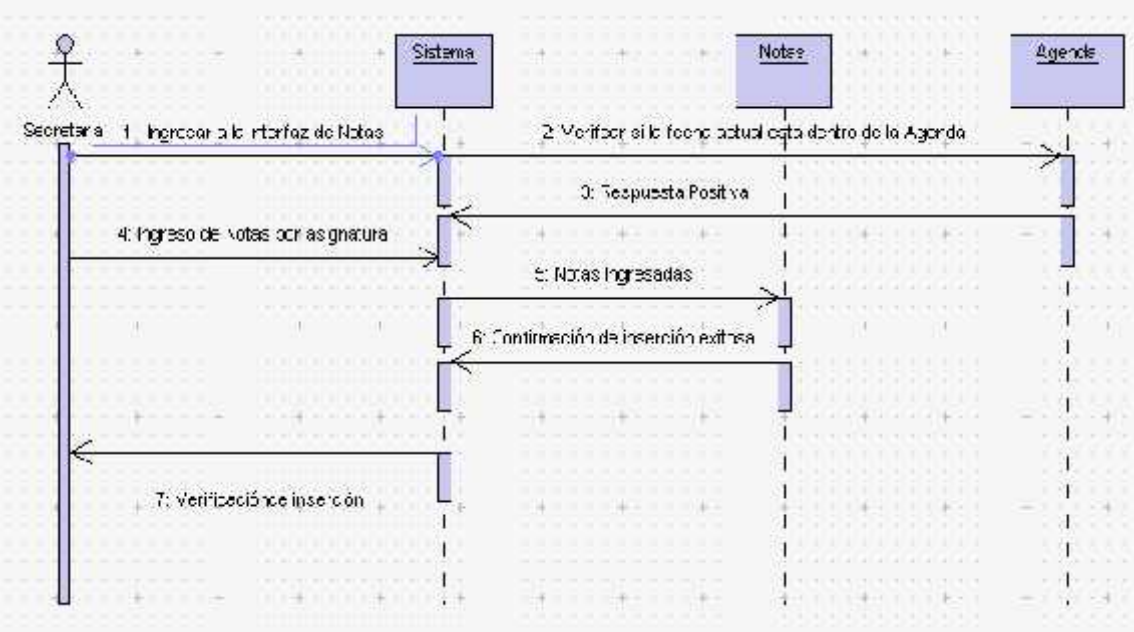


Figura 3.2. Diagrama de secuencia del Flujo Básico de Inserción de Notas

Flujos Alternativos

Estos son los posibles flujos que se generarían en caso de que algún paso del flujo básico falle, para nuestra metodología omitiremos los flujos alternativos propios de las funcionalidades básicas (datos mal ingresados, errores de integridad referencial por ejemplo: claves repetidas en la inserción) y consideraremos sólo los errores propios de las funcionalidades extras en nuestro ejemplo sería:

El él paso 3: el sistema envía una respuesta negativa, es decir la fecha está fuera de la agenda entonces debemos hacer lo siguiente:

3.1. Solicitar autorización de la autoridad competente para realizar la modificación.

3.2. Se ingresa la autorización

3.3. Se registra la autorización y se prosigue.

Estos flujos también se pueden representar en diagramas de secuencia como se ve en la figura 3.3.

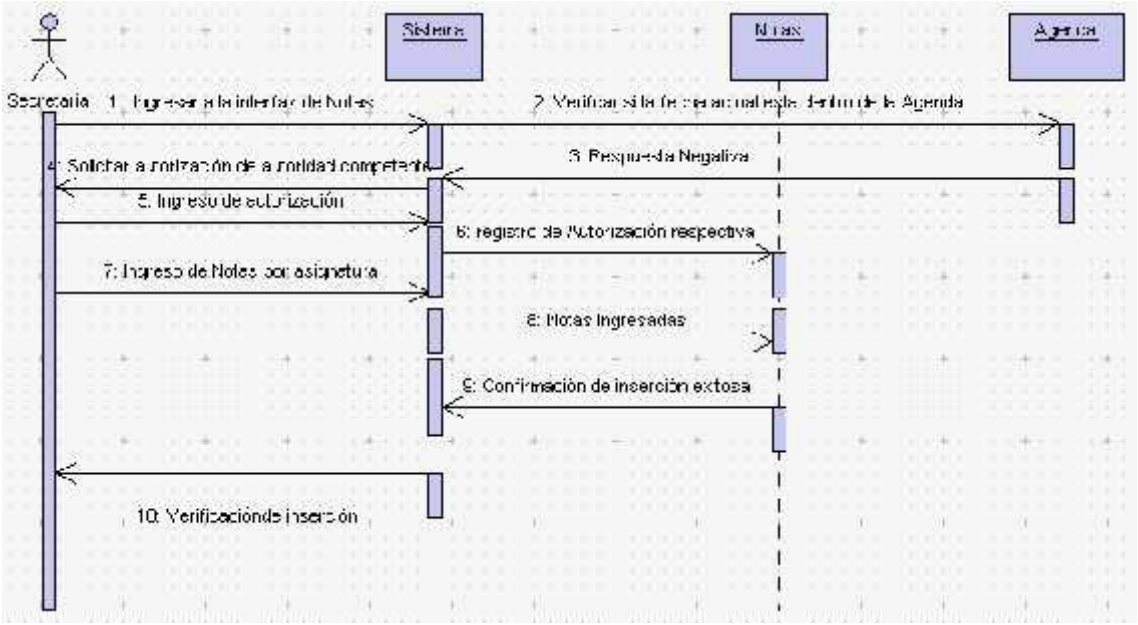


Figura 3.3. Diagrama de secuencia del Flujo Alternativo de Inserción de Notas

Precondiciones

Es el estado previo a la realización de una transacción de algún caso de uso, puede incluir estados del sistema, fechas o cualquier parámetro relevante para que la transacción pueda ser ejecutada por ejemplo

- La secretaria debe haberse identificado exitosamente con usuario y password.

Post condiciones

Son las condiciones en las cuales el sistema queda después de la ejecución de una transacción por ejemplo:

- El sistema una vez ingresadas las notas estas quedan debidamente almacenados en la base de datos.

- Se genera un registro en los logs del sistema indicando la transacción realizada, fecha y responsable

Un Caso de Uso puede tener diferentes funcionalidades Extras y cada una de las debe tener un solo flujo básico y puede tener varios flujos alternativos.

Estas funcionalidades quedarán de esta manera definidas y deben ser presentadas a los usuarios para determinar si estas han capturado lo que él desea del sistema a construir o si se necesitan modificaciones.

Si el equipo de desarrolladores (usualmente uno o dos) no ha capturado los requisitos de los usuarios de una forma satisfactoria deberá ejecutar el paso siguiente que es la documentación de los requisitos obtenidos, en los requisitos que no se haya llegado a una completa definición se debe regresar al paso uno, es decir nuevamente tratará de entender el problema e iniciará un nuevo ciclo para capturar de forma satisfactoria los requisitos de los usuarios.

3.1.1.3. Generación de la documentación de Especificación de requisitos de software.

En este momento debemos plasmar todo los requisitos encontrados en un documento que guardará la especificaciones de requisitos para esto utilizaremos el documento contenido en el anexo 3.

Este paso no es un paso final, pues en la determinación de funcionalidades se puede decidir que los requerimientos que no están suficientemente especificados regresen al

paso de construcción del diagrama de caso de uso general regresando al ciclo que define esta metodología.

Cuando ningún requisito necesite regresar a ser estudiado y nuevamente definido entonces este será el paso final de la especificación de requisitos, es necesario recordar que los requisitos que han sido descritos satisfactoriamente y aceptados por los usuarios ya pueden pasar a la fase de análisis y diseño, es decir no debemos esperar a que todos los requisitos sean aceptados para poder continuar.

CAPÍTULO IV

MARCO HIPOTÉTICO Y METODOLÓGICO

4.1 OBJETO DE ESTUDIO.

El objeto de estudio de la presente investigación es la Propuesta metodológica para especificación de requisitos de software en proyectos pequeños y medianos orientados a objetos, con el objetivo de alcanzar un software que satisfaga las necesidades de los clientes y que sea usado por los mismos para sus labores diarias y no se conviertan en tan sólo un requisito para obtener un certificado de prácticas o para lograr la graduación en la escuela de Ingeniería en Sistemas de la Universidad Estatal de Bolívar.

Se espera que esta metodología sea utilizada en el desarrollo de proyectos pequeños y medianos no sólo en la Universidad Estatal de Bolívar sino en las instituciones y empresas del entorno local o nacional

4.2 DISEÑO DE LA INVESTIGACIÓN.

Se manipulará deliberadamente la variable independiente para analizar su efecto con la variable dependiente por tal motivo el diseño de investigación es CUASI-EXPERIMENTAL.

Para el desarrollo de la investigación se seguirá los siguientes pasos:

- Planteamiento del problema.
- Determinación de los Objetivos de la Investigación.
- Justificación de la Investigación.
- Elaboración del Marco teórico.
- Planteamiento de la Hipótesis.
- Selección del diseño apropiado de investigación.
- Determinación del universo de estudio y determinación de la muestra.
- Recolección de datos.
- Análisis de datos.
- Establecimiento de la propuesta metodológica para especificación de requisitos de software en proyectos pequeños y medianos orientados a objetos,
- Evaluación de la Propuesta metodológica para especificación de requisitos de software en proyectos pequeños y medianos orientados a objetos.
- Prueba de hipótesis
- Conclusiones y recomendaciones del trabajo de investigación.

4.3 SISTEMA DE HIPÓTESIS.

4.3.1 HIPÓTESIS.

H_i: La aplicación de una metodología para especificación de requisitos en proyectos pequeños y medianos; mejorará la productividad de software en los proyectos de tesis y Pasantías en la Universidad Estatal de Bolívar.

H_0: La aplicación de una metodología para especificación de requisitos en proyectos pequeños y medianos; no mejorará la productividad de software en los proyectos de tesis y Pasantías en la Universidad Estatal de Bolívar.

4.3.2 DETERMINACIÓN DE LAS VARIABLES.

Variable Independiente: metodología para especificación de requisitos en proyectos pequeños y medianos

Variables Dependientes: Productividad

4.3.3 OPERACIONALIZACIÓN CONCEPTUAL DE LAS VARIABLES.

VARIABLES		CONCEPTOS
INDEPENDIENTE	Metodología para especificación de requisitos	Resultado del la integración de metodologías de especificación de requisitos.
DEPENDIENTE	Productividad	La relación entre la cantidad de bienes y servicios producidos y la cantidad de recursos utilizados.

TABLA 4.1: Operacionalización Conceptual de las variables

4.3.4 OPERACIONALIZACIÓN METODOLÓGICA DE LAS VARIABLES.

VARIABLES		DIMENSIONES	INDICADORES	INSTRUMENTOS
INDEPEND	Metodología para especificación de requisitos	1. Software	1. Herramientas CASE que permitan formalizar requisitos de software	• Encuestas • Observación
		2. Metodología	2. Actividades a seguir que permitan especificar los requerimientos de software.	• Encuestas • Observación
DEPENDIENTE	Productividad	1. Control de versiones de ERS	3. ERS desarrollados 4. Actualización de requisitos de software.	• Encuestas • Observación
		2. Gestión de requisitos	5. Número de requisitos generados 6. Número de requisitos aceptados	• Encuestas • Observación
		3. Gestión de documentación	7. Control de cambios en requisitos 8. Control de versiones de ERS.	• Encuestas • Observación

TABLA 4.2: Operacionalización Metodológica de las variables

4.4 POBLACIÓN Y MUESTRA.

POBLACIÓN.- La población está constituida por los alumnos que realizan prácticas pre - profesionales y los egresados que están elaborando la tesis de grado de la Escuela de Sistemas de la Facultad de Ciencias Administrativas Gestión Empresarial e Informática de la Universidad Estatal de Bolívar y por los directores de los departamentos de las diferentes instituciones para las cuales se han desarrollado software.

MUESTRA.- En el caso a investigarse se tomará a todo el universo de los alumnos que se encuentran realizando desarrollo de software en la Escuela de Sistemas de la Facultad de Ciencias Administrativas Gestión

Empresarial e Informática de la Universidad Estatal de Bolívar y a todos los directores departamentales para los cuales se ha desarrollado software.

4.5 MÉTODOS, TÉCNICAS E INSTRUMENTOS.

4.5.1 TIPO DE INVESTIGACIÓN.

La investigación a realizarse es de tipo DESCRIPTIVA, APLICATIVA.

Descriptiva: Se describirán los pasos y buenas prácticas a seguir para llegar a generar un conjunto de requisitos de software que representen lo que los usuarios finales quieren del software a desarrollarse.

Aplicativa: Se deberán aplicar los pasos descritos en los desarrollos de software que se ejecuten como tesis de grado y resultados de las prácticas pre – profesionales, los mismos que permitirán aplicar la metodología para especificación de requisitos en proyectos pequeños y medianos

4.5.2 INSTRUMENTOS DE RECOLECCIÓN DE DATOS.

Se aplicarán encuestas para la recolección de información dirigida a los alumnos y entrevistas dirigidas a los directores de los departamentos para los cuales se han desarrollado software, con estos instrumentos queremos determinar si los requisitos de software han sido claros, entendibles y evaluables en los proyectos de software con y sin el uso la metodología para especificación de requisitos en proyectos pequeños y medianos, Los formatos de los cuestionarios se encuentran en el anexo 2.

Dichos cuestionarios están enfocados en los siguientes aspectos:

- **Determinación del conocimiento de metodología para la especificación de requisitos de software orientados a objetos y su aplicación en los proyectos de tesis y prácticas pre profesionales:** dirigido los estudiantes que realizan práctica pre - profesionales y egresados de la Escuela de Ingeniería de Sistemas que están realizando tesis de grado que involucran desarrollo de software.

- **Evaluación del software desarrollado en las prácticas pre - profesionales y tesis de grado para determinar si estos han satisfecho los requisitos de los usuarios finales:** Se aplicará a los directores departamentales de las diferentes instituciones para las cuales se ha desarrollado software.

4.5.2.1 VALIDACIÓN DE INSTRUMENTOS

Los instrumentos se han validado con el asesoramiento de la tutora de tesis quien tiene experiencia en la elaboración de estos instrumentos, además se realizó una muestra aplicando esta a un grupo reducido de la población para validar los instrumentos

4.5.3 TÉCNICAS PARA COMPROBACIÓN DE HIPÓTESIS

La técnica escogida es la de chi cuadrado que es una prueba no paramétrica, que nos sirve cuando se desea evaluar la dependencia de dos variable cualitativas, esto se puede decir con un nivel de confianza que se fija previamente.

Para calcular el X^2 es necesario calcular las frecuencias esperadas, es decir aquellas que deberían haberse observado si las variables fuesen independientes, y compararlas con las frecuencias observadas, para esto se aplica la siguiente fórmula:

$$i^2 = \sum_{i=1}^{r} \sum_{j=1}^{k} \frac{\left(O_{ij} - E_{ij}\right)^2}{E_{ij}}$$

Donde:

O_{ij} denota a las frecuencias observadas en la fila i y la columna j

E_{ij} denota a las frecuencias esperadas o teóricas aquella frecuencia que se observaría si ambas variables fuesen independientes.

El objetivo es medir la diferencia entre el valor que debería existir si las variables fueran independientes versus el valor real observado, a mayor diferencia mayor es la dependencia de las variables estudiadas

El test X^2 es así un test no dirigido (test de planteamiento bilateral), que nos indica si existe o no relación entre dos factores pero no en qué sentido se produce tal asociación.

Para obtener los valores esperados E_{ij}, estos se calculan a través del producto de los totales marginales dividido por el número total de casos (n). Para el caso más sencillo de una tabla 2x2 , se tiene que:

$$E_{11} = \frac{(a+b)\times(a+c)}{n} \qquad E_{21} = \frac{(c+d)\times(a+c)}{n}$$

$$E_{12} = \frac{(a+b)\times(b+d)}{n} \qquad E_{22} = \frac{(c+d)\times(b+d)}{n}$$

El valor de X^2 se distribuye en una distribución conocida denominada chi cuadrado, esta distribución tiene un parámetro denominado grado de libertad (G.l.) este parámetro se calcula con la siguiente fórmula:

G.I. =(r-1)(k-1)

Donde:

r es el número de filas

k es el número de columnas

Una vez determinado los grados de libertad se debe determinar α que es un parámetro que determina la seguridad, en dependencia de estos dos parámetros se tiene el valor de X^2

Para α =0.01 con un GI=1 X^2= 6,63

Para α =0.05 con un GI=1 X^2 =3,84

Estos valores se encuentran en las tablas de distribuciones de la función chi cuadrado (ver anexo 3).

Si la diferencia entre los valores de X^2 obtenida y el valor de X^2 de la tabla de distribución de la función chi cuadrado es muy grande s rechaza la hipótesis H_0 y se acepta la hipótesis alternativa H_i

CAPÍTULO V

ANÁLISIS E INTERPRETACIÓN DE RESULTADOS

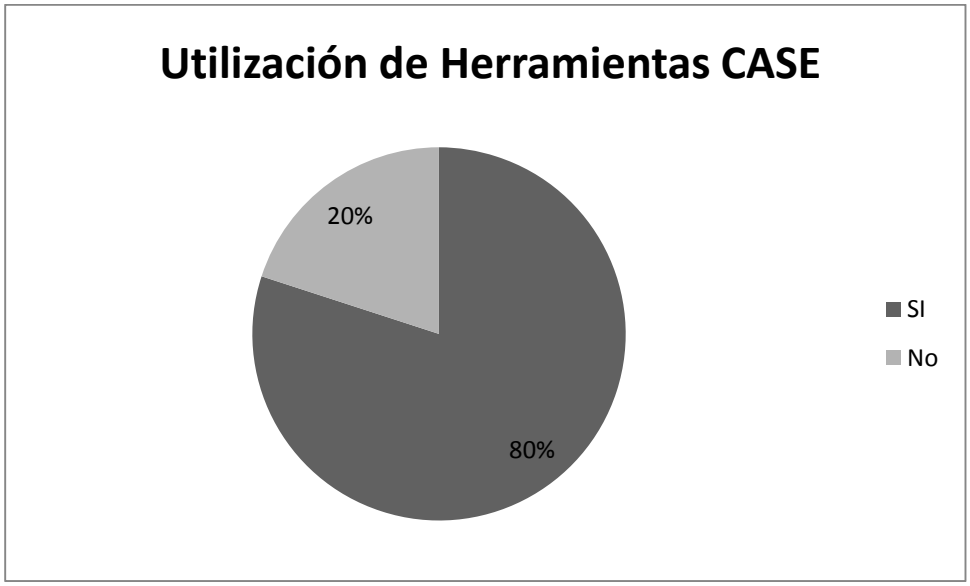
5.1 ANÁLISIS DE LAS DIMENSIONES DE LA VARIABLE INDEPENDIENTE.

En la encuesta aplicada y mediante la observación de la documentación entregada por los desarrolladores de software se puede determinar cuántos de ellos usaron herramientas CASE para la especificación de requisitos de software, así como si se utilizó una metodología o no para establecer los requisitos de software, que son las dimensiones para la variable independiente.

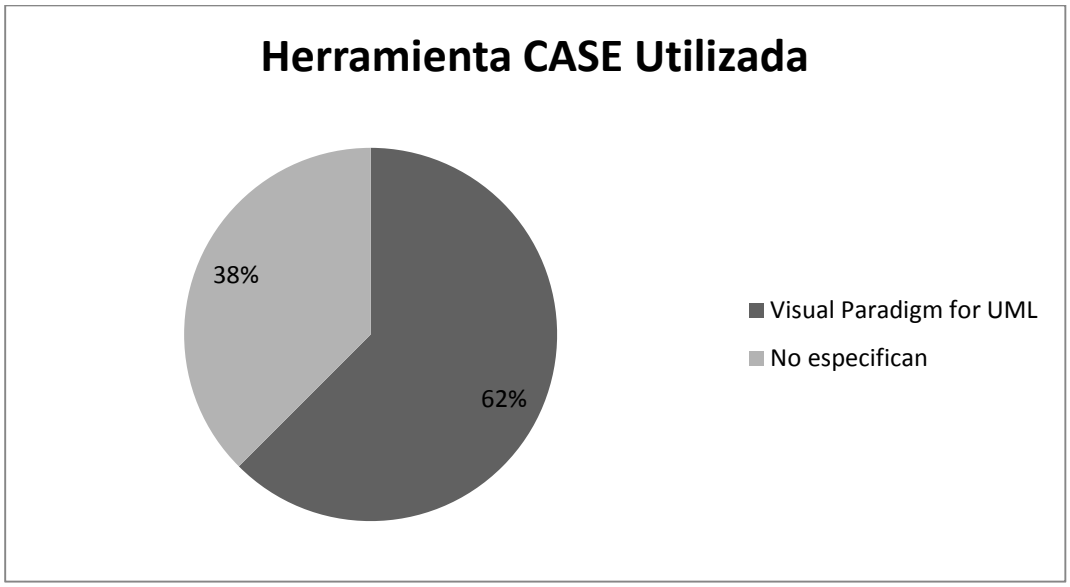
5.1.1. DIMENSIÓN SOFTWARE

Se trata de determinar si se han utilizado herramientas CASE y cuales se han utilizado.

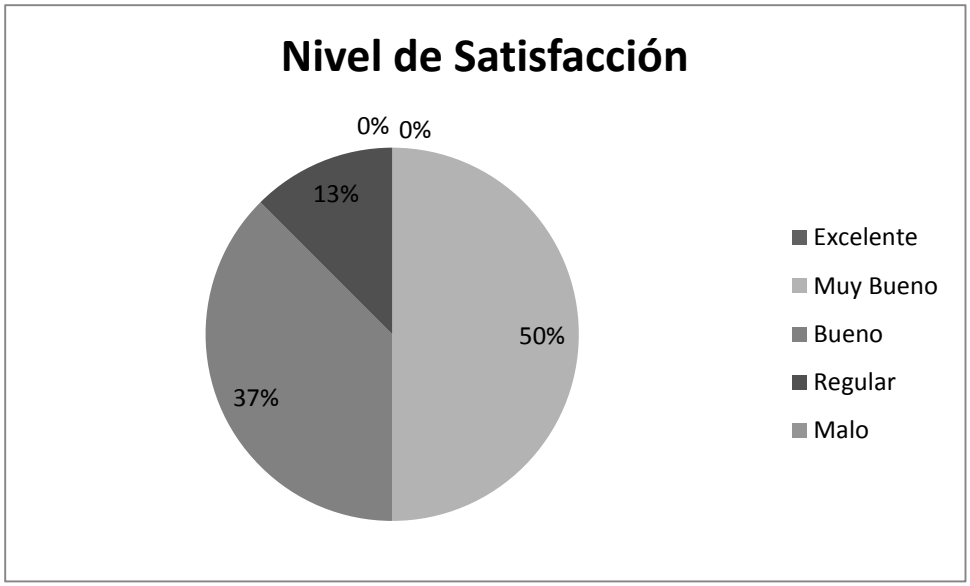
5.1.1.1 Indicador Herramientas CASE utilizadas



Como se puede observar el 80% de los casos han utilizado una herramienta CASE



Un 62% uso la herramienta Visual Paradigm for UML en los otros casos no se especifica que herramienta se utilizó.



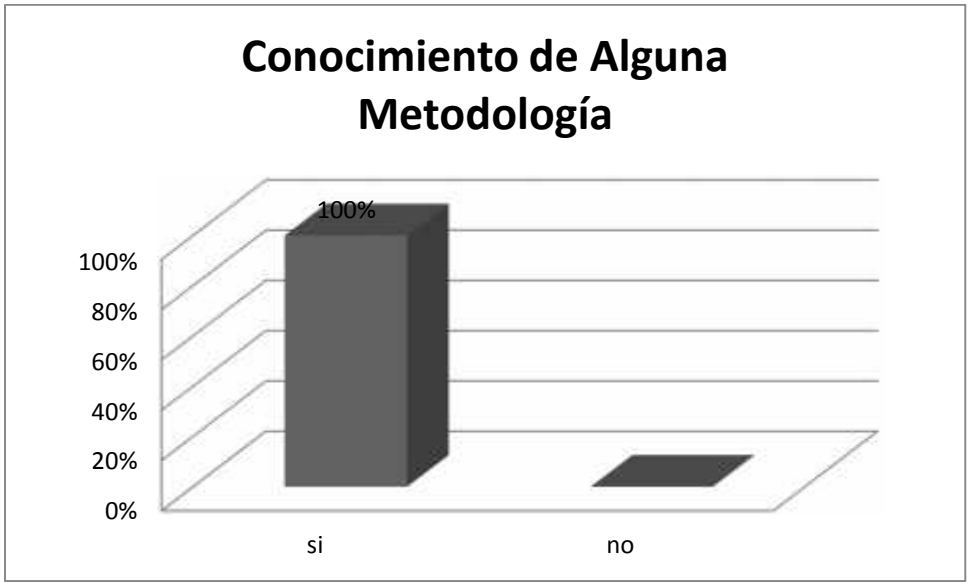
El nivel de satisfacción de utilizar una herramienta Case esta en el rango de Muy Bueno y Bueno en su mayoría con un 87%

Interpretación.- Analizando los resultados anteriores podemos decir que un alto porcentaje de los desarrolladores han utilizado una herramienta CASE, el hecho de que estos se hayan decidido por la herramienta Visual Paradigm for UML es porque los desarrolladores ya han utilizado esta herramienta en el proceso de aprendizaje, la experiencia de utilizar una herramienta CASE ha sido calificada como buena o muy buena en su mayoría debido a la facilidad de construcción de modelos gráficos que estas herramientas proporcionan.

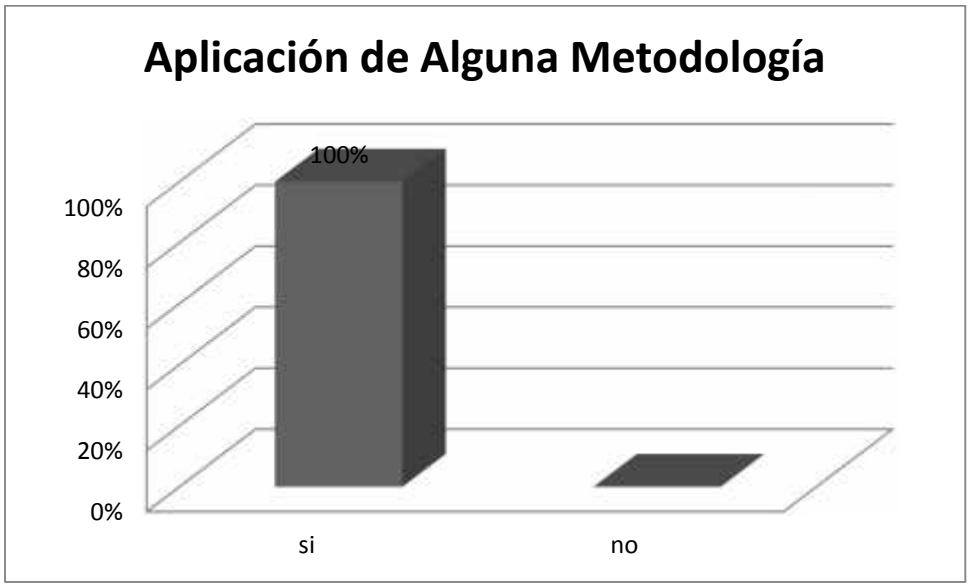
5.1.2 DIMENSIÓN METODOLOGÍA

Se trata de determinar si los desarrolladores conocen y han utilizado una metodología para la especificación de requisitos de software

5.1.2.1 Indicador Seguimiento de las actividades a seguir



El 100% de los desarrolladores conocen alguna metodología de especificación de requisitos de software.



El 100% de los desarrolladores aplicaron alguna metodología de especificación de requisitos de software.

Interpretación.- Todos los desarrolladores dicen conocer y han aplicado alguna metodología de especificación de requisitos de software en el desarrollo de sus sistemas.

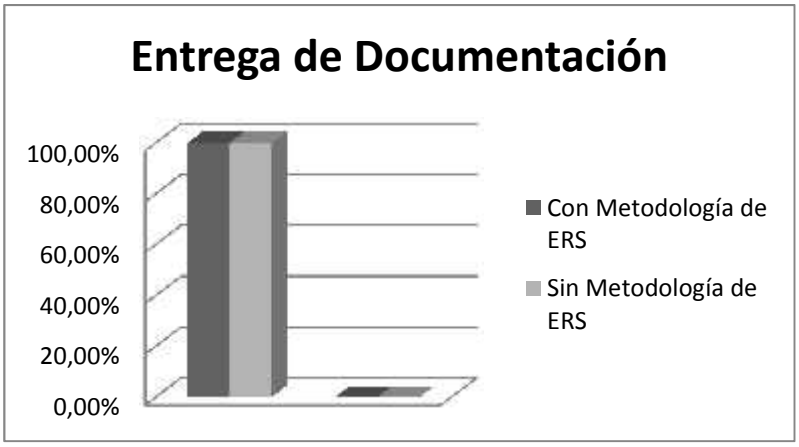
5.2 ANÁLISIS DE LAS DIMENSIONES DE LA VARIABLE DEPENDIENTE.

Para el análisis de la dimensiones de las variables dependiente consideraremos tanto la documentación y las aplicaciones generadas como productos del desarrollado software sin y con la metodología de especificación de requisitos de software.

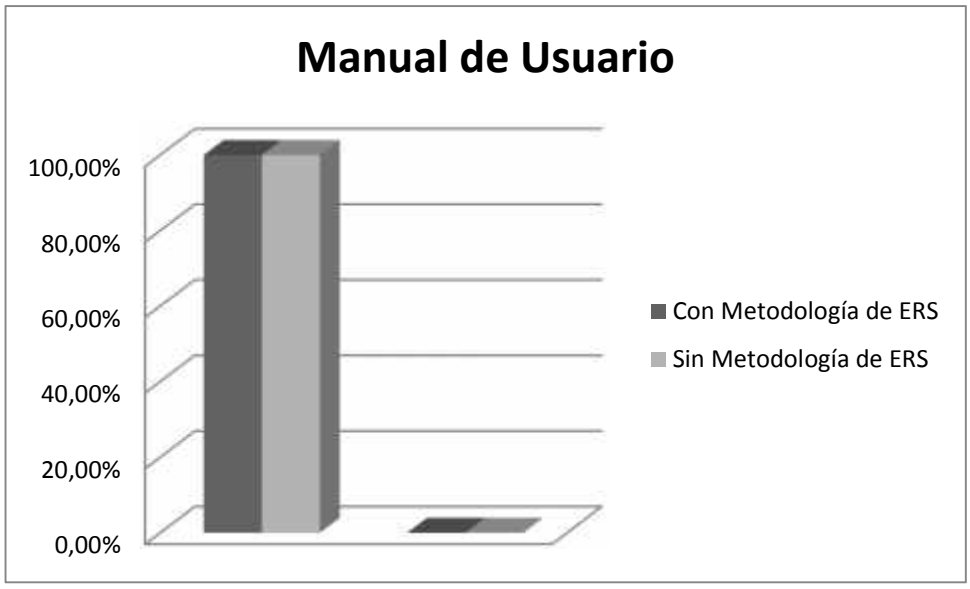
Se cruzara información entre los requisitos que se definieron en los diferentes proyectos con la aceptación que los sistema han tenido para definir el número de requisitos generados, formalizados y aceptados, así como si existió el control de cambios y el control de versiones de ERS.

5.2.1 DIMENSIÓN CONTROL DE VERSIONES DE ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE

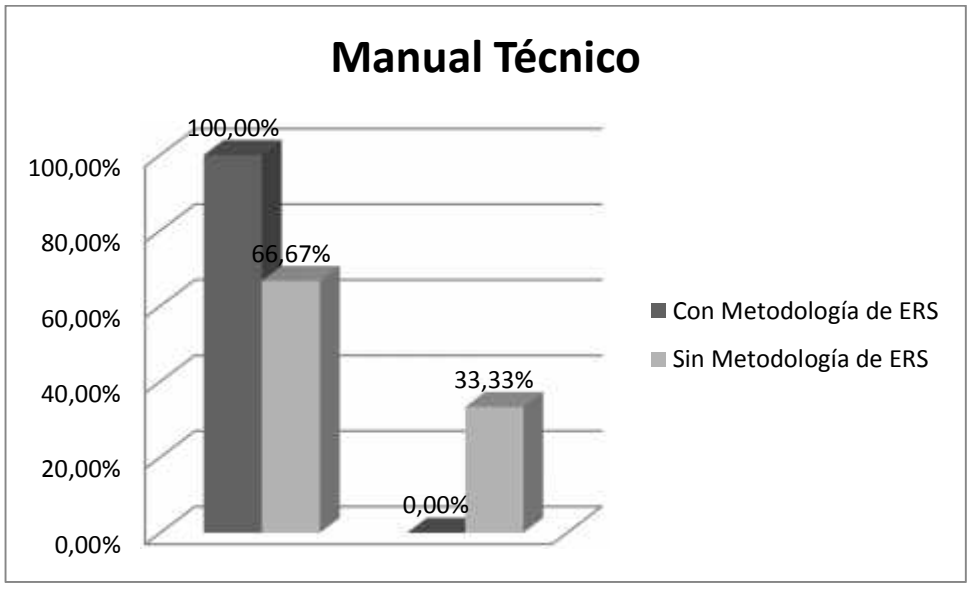
5.2.1.1 INDICADOR GENERACIÓN DE DOCUMENTOS DE REQUISITOS DE SOFTWARE



Todos los proyectos con y sin metodología de Especificación de requisitos de Software entregaron documentación, a continuación veremos qué tipo de documentación se entrego.



Todos los proyectos entregaron manual de usuario.



Todos los proyectos que utilizaron la metodología de especificación de requisitos entregaron manual técnico, un 66.67% de los proyectos que no trabajaron con la metodología no entregaron manual técnico.

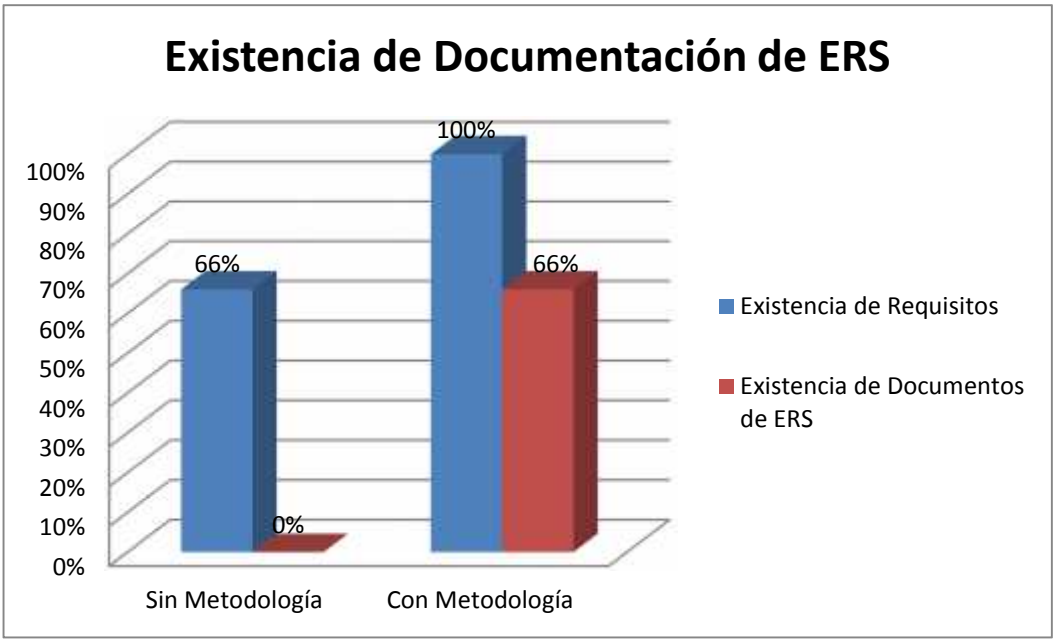
ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE DESARROLLADOS

Revisando la documentación entregada podemos determinar si se ha desarrollado la documentación de Especificación de Requisitos de software de este análisis se desprende la tabla siguiente:

Casos de Estudio	**Existencia de Requisitos**	**Existencia de Documentos de ERS**
Sin Metodología		
Caso 1	Si	No
Caso 2	Si	No
Caso 3	No	No
Con Metodología		
Caso 1	Si	Si
Caso 2	Si	Si
Caso 3	Si	No

Cuantificación

	Sin Metodología	Con Metodología
Existencia de Requisitos	66,00%	100,00%
Existencia de Documentos de ERS	0,00%	66,00%



Interpretación.- En lo que se refiere a la documentación todos los proyectos entregaron algún tipo de documentación como el manual de usuario o el manual técnico, esto es debido a que en la realización de tesis obligatoriamente se debe entregar esta documentación, en la revisión de la documentación entregada se puede ver que en todos los casos que se utilizó la metodología existen enlistados los requisitos, mientras que en los casos que no se utilizó la metodología un 66% tiene un listado de requisitos pero ninguno de ellos formalizada en un documento de Especificación de Requisitos de Software

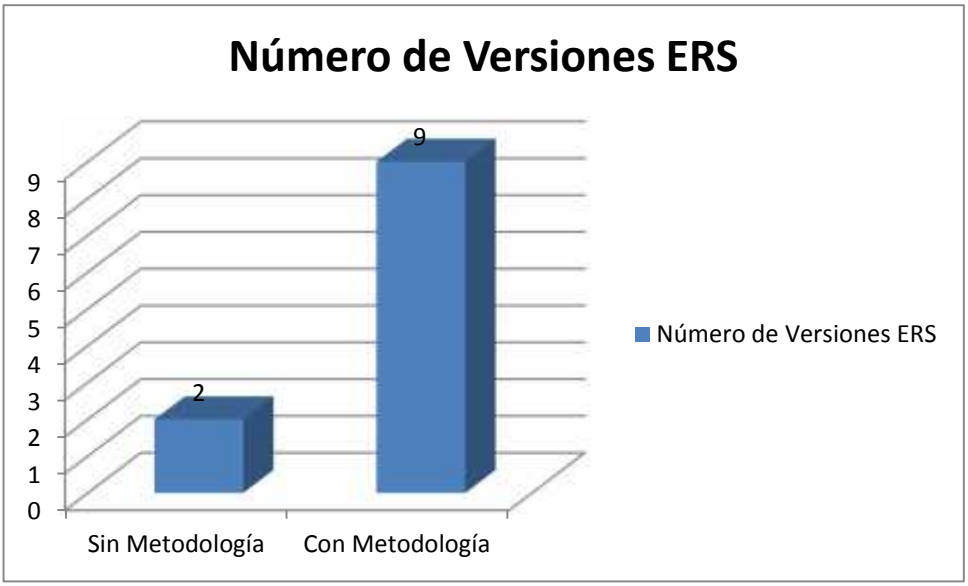
5.2.1.2 INDICADOR ACTUALIZACIÓN DE REQUISITOS DE SOFTWARE

Para cuantificar este indicador revisaremos el número de versiones de documentos de especificación de requisitos, utilizamos el número de versiones de ERS, debido que una vez que se revisaron los requisitos y

se descubrieron nuevos elementos se debe desarrolla una nueva versión.

Casos de Estudio	**Número de Versiones de ERS** Nota: se considera al listado de requisitos como una versión ERS
Sin Metodología	
Caso 1	1
Caso 2	1
Caso 3	0
Con Metodología	
Caso 1	4
Caso 2	3
Caso 3	2

	Número de Versiones ERS
Existencia de Requisitos	2
Existencia de Documentos de ERS	9



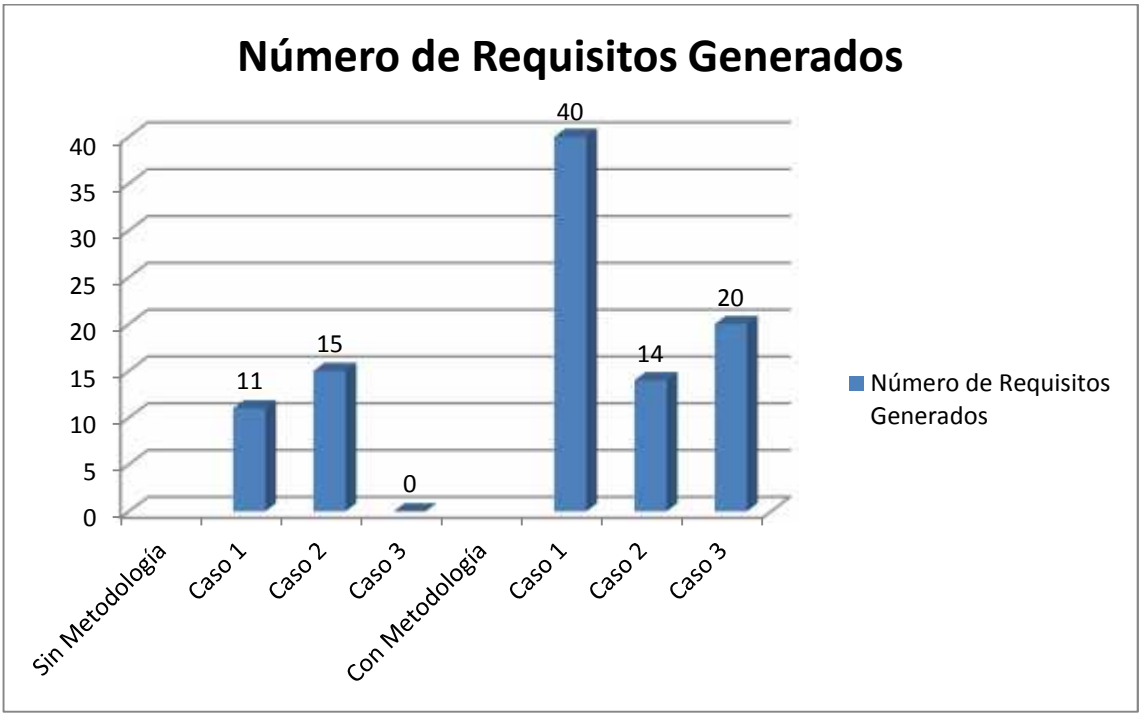
Interpretación.- Los desarrollo que siguieron la metodología propuesta generaron muchas más versiones que los que no utilizaron la metodología, lo que quiere decir que se actualizaron los requisitos y se evidenció esto en la documentación.

5.2.2 DIMENSIÓN GESTIÓN DE REQUISITOS

5.2.2.1 INDICADOR NÚMERO DE REQUISITOS GENERADOS.

Casos de Estudio	**Número de Requisitos Generados**
Sin Metodología	
Caso 1	11
Caso 2	15
Caso 3	0
Con Metodología	
Caso 1	40

Caso 2	14
Caso 3	20



Interpretación.- En la mayoría de casos que usaron la metodología se especificaron mayor número de requisitos que en los que no la usaron, excepto en un caso. Se puede ver el caso extremo en un caso que no utilizó la metodología en el cual no existe ni siquiera un listado de requisitos.

5.2.2.2 INDICADOR NÚMERO DE REQUISITOS ACEPTADOS.

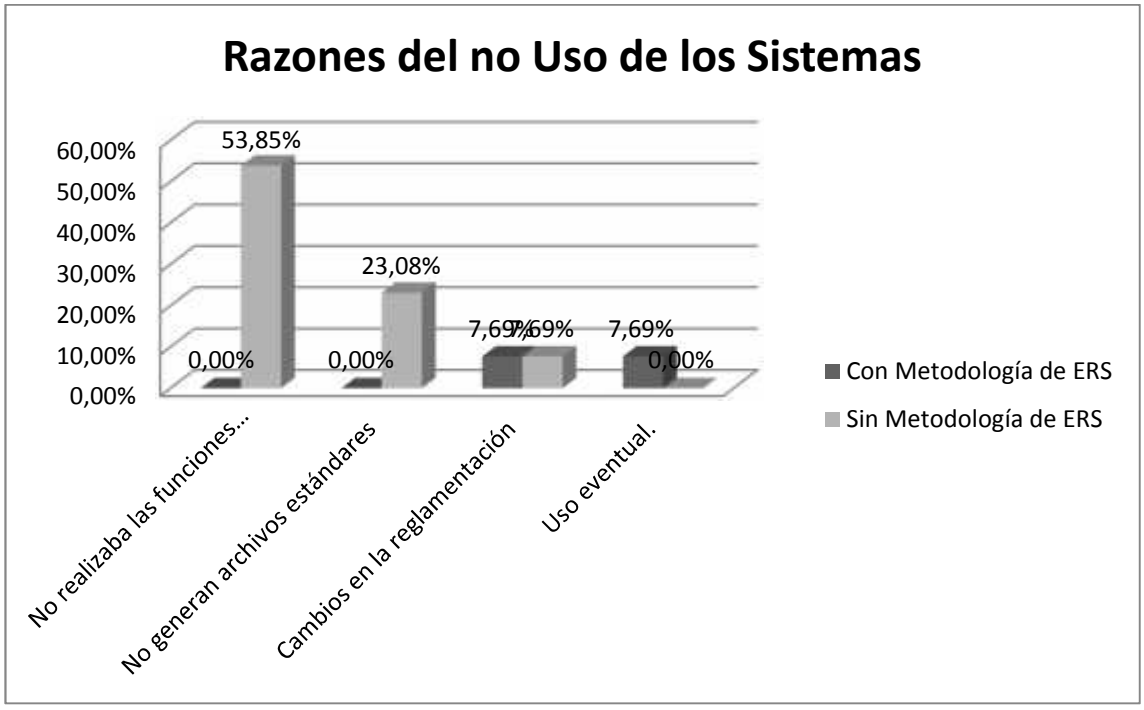
Para determinar el número de requisitos aceptados nos basamos en los requisitos listados en la documentación comparados con el sistema

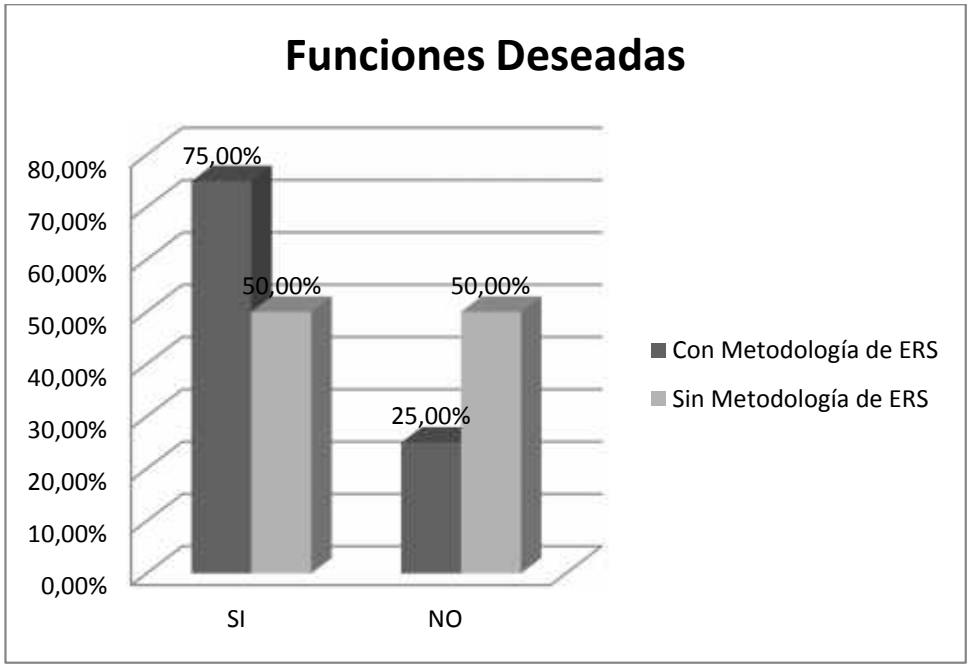
generado, además se considerará si el sistema está en uso y la aceptación de estos sistemas.

Casos de Estudio	**Número de Requisitos Generados**	**Número de Requisitos Aceptados** Nota: Requisitos que se implementaron en el sistema
Sin Metodología		
Caso 1	11	10
Caso 2	15	15
Caso 3	0	No verificable
Con Metodología		
Caso 1	40	40
Caso 2	14	14
Caso 3	20	19

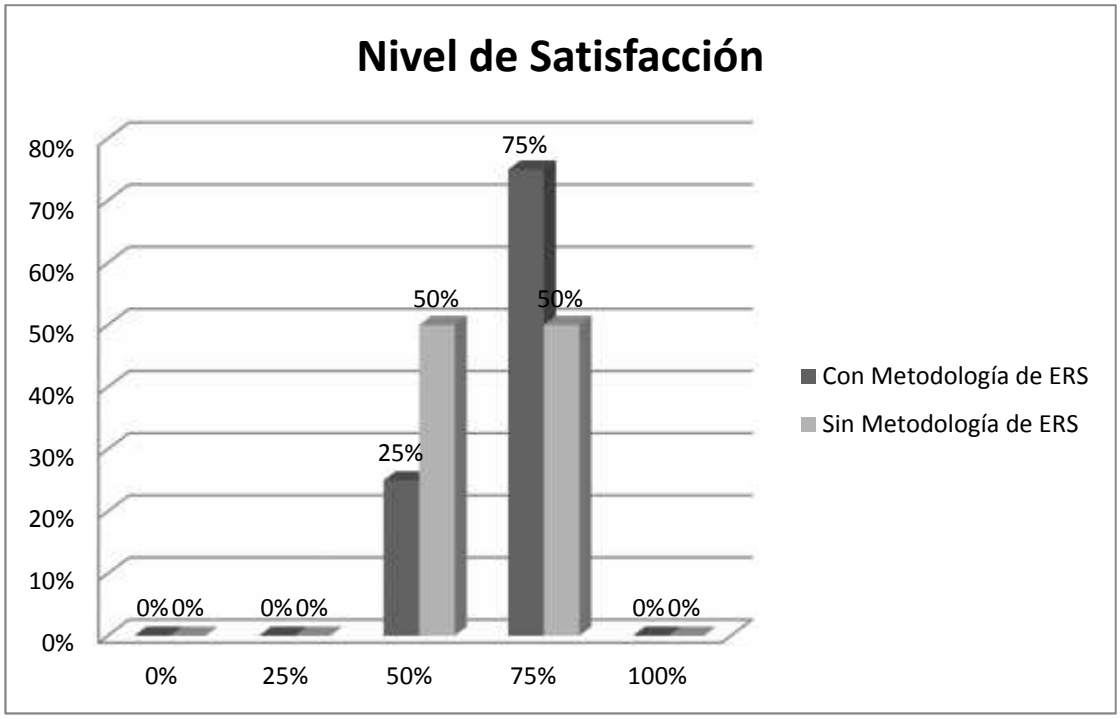
Como podemos observar un gran porcentaje de los sistemas que no utilizaron la metodología no están en uso, entre las razones señaladas para esto tenemos:

- No realizaba las funciones esperadas, es la principal razón
- No generan archivos estándares que son aceptados por otros sistemas.
- Cambios en la reglamentación que no fue adoptada por los sistemas.
- El sistema se usa eventualmente.





Este gráfico representa la apreciación de los usuarios acerca de las funciones que tiene los sistemas que están en uso, es decir si las funciones que tienen los sistemas son las que ellos esperaban.



Interpretación.- Tanto los sistemas desarrollados con la metodología de especificación de requisitos de software para proyectos pequeños y medianos, así como los que no la utilizaron tienen diferencias mínimas entre el número de requisitos generados versus el número de requisitos aceptados, sin embargo el software generado sin la utilización de la metodología tiene una bajo nivel de aceptación y entre las principales razones para esto tenemos que: no realizaban las funciones deseadas y que no generaban archivos estándares, la primera razón son requisitos funcionales y la segunda requisitos no funcionales.

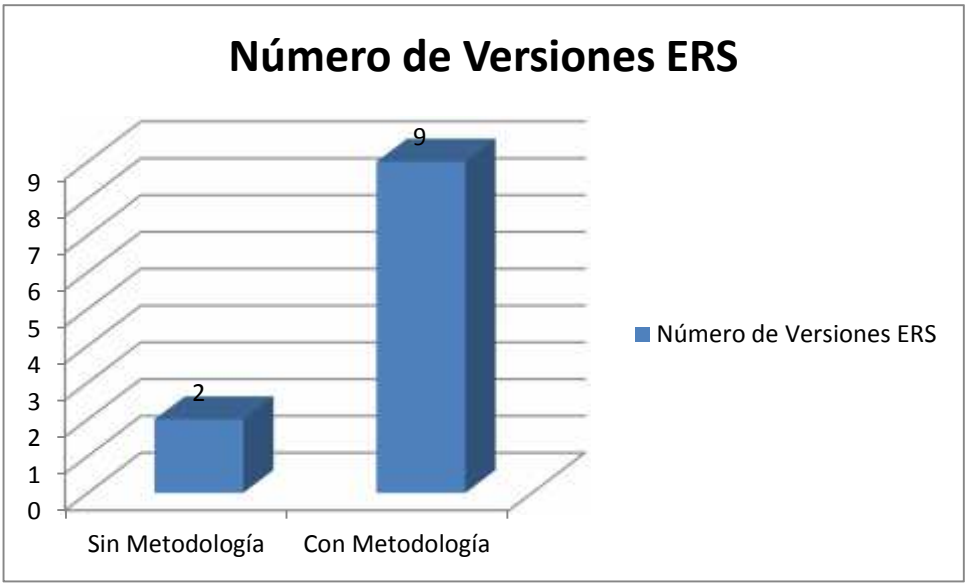
5.2.3 DIMENSIÓN GESTIÓN DE DOCUMENTACIÓN

5.2.3.1 INDICADOR CONTROL DE CAMBIOS EN REQUISITOS.

El control de cambios se verifica con el número de versiones registradas en los documentos de especificación de requisitos, debido a que en de una versión a otra existen cambios, en cada versión del documento se registra, fecha, cambios y responsable entre otros datos, los que permiten hacer un seguimiento a los cambios.

Casos de Estudio	**Número de Versiones de ERS** Nota: se considera al listado de requisitos como una versión ERS
Sin Metodología	
Caso 1	1
Caso 2	1
Caso 3	0
Con Metodología	
Caso 1	4
Caso 2	3
Caso 3	2

	Número de Versiones ERS
Existencia de Requisitos	2
Existencia de Documentos de ERS	9



Interpretación.- Sólo la existencia de versiones de requisitos de software debidamente registradas y documentadas permitirá realizar un control de cambios efectivo, el mayor número de versiones en los desarrollos que siguieron la metodología propuesta demuestra que en estos el seguimientos de cambios se puede realizar.

5.2.3.2 INDICADOR CONTROL VERSIONES DE ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE

Casos de Estudio	**Número de Versiones de ERS** Nota: se considera al listado de requisitos como una versión ERS
Sin Metodología	
Caso 1	1
Caso 2	1
Caso 3	0
Con Metodología	

Caso 1	4
Caso 2	3
Caso 3	2

Interpretación.- Sólo en los sistemas que siguieron la metodología propuesta se puede realizar un control de versiones, en los casos que no utilizaron la metodología propuesta esto no posible debido a que sólo existe en estos un listado de requisitos.

5.3 COMPROBACIÓN DE LA HIPÓTESIS.

Se ha tabulado e interpretado los resultados tanto de: la encuesta, la entrevista y la observación de la documentación y el software

Para nuestra investigación hemos identificado una variable independiente que es la Metodología para especificación de requisitos (X) y la dependiente que es Productividad (Y), lo que quiere decir que la variable independiente se puede manipular para observar el comportamiento de la variable dependiente, por tal motivo consideraremos dos variante de X que son:

- X_1 resultados sin la aplicación de la Metodología de especificación de Requisitos Orientada a Objetos.
- X_2 resultados con la aplicación de la Metodología de especificación de Requisitos Orientada a Objetos.

A continuación se construirá una tabla en la que se presentará los índices estudiados estableciendo cómo se comporta la variable Y en relación a las

variables X_1 y X_2. Para la construcción de esta tabla consideraremos los diferentes valores obtenidos de la tabulación de resultados y observación en los diferentes indicadores para los proyectos con y sin el uso de la Metodología de Especificación de Requisitos de Software

VARIABLE DEPENDIENTE PRODUCTIVIDAD		**VARIABLE INDEPENDIENTE** Metodología para especificación de requisitos			
		SIN Metodología para especificación de requisitos		**CON** Metodología para especificación de requisitos	
DIMENSIONES	**INDICADORES**	**MEJORA**	**NO MEJORA**	**MEJORA**	**NO MEJORA**
	ERS desarrollados	**0**	**100**	**77**	**33**
Control de versiones de ERS	Actualización de requisitos de software.	33	77	100	0
Gestión de requisitos	Número de requisitos generados	33	77	77	33
	Número de requisitos aceptados	96	4	98	2
Gestión de documentación	Control de cambios en requisitos	0	100	100	0
	Control de versiones de ERS.	0	100	100	0
TOTALES		162	458	552	68

Con los resultados obtenidos en la tabla anterior se genera la tabla de cruzada siguiente:

PRODUCTIVIDAD	Metodología para especificación de requisitos		
	SIN	**CON**	**TOTAL**
Mejora la Productividad	162,00	552,00	714,00
No mejora la Productividad	458,00	68,00	526,00
TOTAL	620,00	620,00	1240,00

A continuación se calculan las frecuencias esperadas, la cual constituye la tabla que esperaríamos encontrar si las variables fueran estadísticamente independientes o no estuvieran relacionadas, para construir esta tabla aplicaremos la fórmula siguiente:

$$fe = \frac{(total\_de\_fila)(total\_de\_columna)}{N}$$

Donde:

N es el número total de frecuencias observadas

fe es la frecuencia esperada

Obteniendo la siguiente tabla:

PRODUCTIVIDAD	Metodología para especificación de requisitos		
	SIN	**CON**	**TOTAL**
Mejora la Productividad	357,00	357,00	714,00
No mejora la Productividad	263,00	263,00	526,00
TOTAL	620,00	620,00	1240,00

En las tablas anteriores se tienen las frecuencias observadas es decir las que se han deducido de la investigación estadística y las frecuencias esperadas son las frecuencias que se esperarían si la distribución de errores no dependiera de la utilización de la Metodología de Especificación de requisitos de Software Orientada a Objetos, por lo tanto podemos calcular el valor estadístico de X^2 el mismo que se calcula con la fórmula:

$$X^2 = \sum_{i=1}^{r}\sum_{j=1}^{k}\frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

Donde

O_{ij} denota a las frecuencias observadas en la fila i de la columna j.

E_{ij} denota a las frecuencias esperadas en la fila i de la columna j.

La aplicación de la fórmula anterior se puede ver en la siguiente tabla:

CELDA	O	E	O-E	$(O-E)^2$	$(O-E)^2/E$
Mejora/sin ERS	162,00	357,00	-195,00	38025,00	106,51
Mejora/con ERS	552,00	357,00	195,00	38025,00	106,51
No Mejora/sin ERS	458,00	263,00	195,00	38025,00	144,58
No Mejora/con ERS	68,00	263,00	-195,00	38025,00	144,58
				X^2	502,19

Una vez obtenidos estos resultados vamos a utilizar la hipótesis nula y la hipótesis alternativa que son:

Hipótesis Nula

H_0: La aplicación de una metodología para especificación de requisitos en proyectos pequeños y medianos; no mejorará la productividad de software en los proyectos de tesis y Pasantías en la Universidad Estatal de Bolívar.

Hipótesis Alternativa

H_i: La aplicación de una metodología para especificación de requisitos en proyectos pequeños y medianos; mejorará la productividad de software en los proyectos de tesis y Pasantías en la Universidad Estatal de Bolívar.

Utilizando la hipótesis nula trabajaremos utilizando el parámetro de grado de libertad (GI) que para el de una tabla de contingencia de r filas y k columnas es:

GI = (r-1)(k-1)

Para una tabla 2x2 el GI es igual a 1.

Tomando el valor de la tabla de distribución (anexo 3) del chi – cuadrado con un grado de independencia de 1 y para una seguridad del 95% tomaremos el valor de 0.05, entonces el valor teórico de la distribución chi – cuadrado es de 3,84

Interpretación

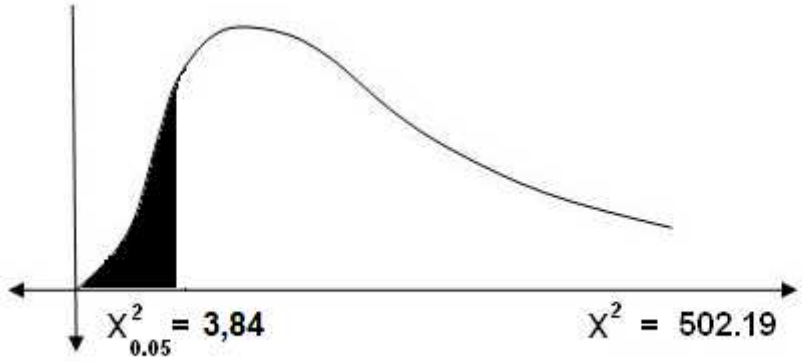
Variable Independiente

Propuesta metodológica para especificación de requisitos de software en proyectos pequeños y medianos orientados a objetos

Variable Dependiente

Productividad

$$X^2_{0.05} = 3{,}84$$



Como se puede observar el valor obtenido de X^2 para la variable dependiente es de 206.73, que es muy superior al valor $X^2_{0.05} = 3{,}84$ que se obtuvo de la tabla de distribución con un grado de libertad de 1, por lo tanto podemos concluir que las dos variables no son independientes, si no que están asociadas y como consecuencia de estos podemos rechazar la Hipótesis Nula y por lo tanto aceptamos se acepta la Hipótesis Alternativa que es la de la investigación.

Por lo tanto:

La aplicación de una metodología para especificación de requisitos en proyectos pequeños y medianos; mejorará la productividad de software en los proyectos de tesis y Pasantías en la Universidad Estatal de Bolívar.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- En base a los resultados obtenidos se ha evidenciado la necesidad de la aplicación de una metodología práctica que permita gestionar de mejor forma los requisitos de software, haciendo que los sistemas sean realmente usados.

- El control de cambios y el contacto constante con los usuarios finales permite refinar los requisitos de software mejorando la aceptación que tendrán los sistemas desarrollados.

- Un 58,85% de los sistemas dejaron de usarse porque no realizaban las funciones para las cuales fueron desarrollados, evidenciando una deficiente especificación de requisitos de software cuando no se ha utilizado una metodología práctica.

- Un 75% de los sistemas que fueron desarrollados siguiendo la metodología para la especificación de requisitos, realizan las funciones que los usuarios finales deseaban con un nivel de satisfacción de un 75%,

- A pesar de que todos los desarrolladores tenían conocimiento de alguna metodología para la especificación de requisitos de software y la aplicaron el nivel de aceptación de los sistemas es de un 50% para los sistemas que no aplicaron la metodología propuesta.

- Un 80% de los desarrolladores usaron una herramienta CASE para la construcción de modelos gráficos, el 87% de estos desarrolladores califican el uso de esta herramienta entre bueno y muy bueno.

RECOMENDACIONES

- Los docentes tutores de las tesis y directores de las pasantías de la Escuela de Sistemas de la Universidad de Bolívar, tienen que constatar que los alumnos generen la documentación de requisitos de software y verificar que los mismos sean evaluados con los usuarios finales.

- En las asignaturas que se imparten a los alumnos de la Escuela de Ingeniería en Sistemas se debe impartir la Metodología Propuesta para que los alumnos puedan aplicar la misma en sus proyectos de fin de ciclo y de esta manera se familiaricen con esta.

- Se debería utilizar diferentes herramientas CASE en el desarrollo de proyectos de fin de ciclo, estas herramientas deben estar orientadas a diferentes aspectos del desarrollo de software y no sólo para que sean usadas como graficadores.

- Los sistemas desarrollados por los alumnos de la Escuela de Sistemas deben tener un seguimiento, es decir, una vez que los sistemas ha sido implantados en una institución, se debe estar en constante contacto para realizar mantenimiento y actualización de los sistemas, para evitar de esta manera que estos se dejen de usar por cambios de reglamentación o falta de implementación defunciones adicionales.

GLOSARIO

ERS: Especificación de Requisitos de Software

CASE: Ingeniería de Software Asistida por Computador

RUP: Rational Unified Process

UML: Unified Modeling Language

DFD: Diagrama de Flujo de Datos

SREM: Software Requirements Engineering Methodology (Metodología de la Ingeniería de Requerimientos de Software)

RSL: Requirements Engineering Language (Lenguaje de enunciados de requerimientos)

REVS: Requirements Engineering Validation System (Sistema de Validación de Requerimientos)

EIA: Electronic Industries Alliance

ANSI: American National Standards Institute

SyRS : System Requirements Specification (Requerimientos del Sistema).

IEEE

CMM: Capability Maturity Model (Modelo de Capacidad y Madurez)

SEI: Software Engineering Institute de la Universidad Carnegie-Mellon **.**

KPA: Key Process Area

SRI: Servicio de Rentas Internas

BIBLIOGRAFÍA

BIBLIOGRAFÍA GENERAL

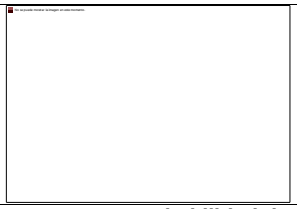
- SENN JAMES A, Análisis y Diseño de Sistemas de Información, Editorial Mc Graw Hill. Primera Edición, 1990.
- ROGER PRESSMAN, Ingeniería del Software.
- MERLIN DORFMAN AND RICHARD H THAYER, Software Engeneering, 1997.
- KENDALL RICHARD, Análisis y Diseño de Sistemas, Editorial Mc Graw Hill, Primera Edición, 1995.
- LAWRENCE PFLEEGER SHARI, Ingeniería de Software Teoría y Práctica, Editorial Prentice Hall, Primera Edición 2002.

BIBLIOGRAFÍA ESPECÍFICA

- RUMBAUGH JAMES; JACOBSON IVAR; BOOCH, GRADY, El Lenguaje Unificado de Modelado. Manual de Referencia, Editorial Pearson Addison Wesley. Segunda Edición, 2004.
- Rational Software White Paper TP026B, Rev 11/01
- ANSI/EIA-632-1998 Approved: January 7, 1999
- IEEE Std 1233, Edición 1998,
- LOPEZ ILLESCAS DIANA CRISTINA, PEÑAHERRERA AROCA ANGELA MARIA, RODRÍGUEZ VEINTIMILLA LOURDES IVON. DESARROLLO DE SOFTWARE UTILIZANDO LA METODOLOGÍA RUP (Process Unified Rational) Caso Práctico: “Sistema Escolástico Parametrizable Latacunga, Agosto del 2004

ANEXOS

Anexo 3

Tabla de Distribución del Chi Cuadrado					
					
	Probabilidad de un valor superior				
Grados de libertad	**0,1**	**0,05**	**0,025**	**0,01**	**0,005**
1	2,71	3,84	5,02	6,63	7,88
2	4,61	5,99	7,38	9,21	10,60
3	6,25	7,81	9,35	11,34	12,84
4	7,78	9,49	11,14	13,28	14,86
5	9,24	11,07	12,83	15,09	16,75
6	10,64	12,59	14,45	16,81	18,55
7	12,02	14,07	16,01	18,48	20,28
8	13,36	15,51	17,53	20,09	21,95
9	14,68	16,92	19,02	21,67	23,59
10	15,99	18,31	20,48	23,21	25,19
11	17,28	19,68	21,92	24,73	26,76
12	18,55	21,03	23,34	26,22	28,30
13	19,81	22,36	24,74	27,69	29,82
14	21,06	23,68	26,12	29,14	31,32
15	22,31	25,00	27,49	30,58	32,80
16	23,54	26,30	28,85	32,00	34,27
17	24,77	27,59	30,19	33,41	35,72
18	25,99	28,87	31,53	34,81	37,16
19	27,20	30,14	32,85	36,19	38,58
20	28,41	31,41	34,17	37,57	40,00
21	29,62	32,67	35,48	38,93	41,40
22	30,81	33,92	36,78	40,29	42,80
23	32,01	35,17	38,08	41,64	44,18
24	33,20	36,42	39,36	42,98	45,56
25	34,38	37,65	40,65	44,31	46,93
26	35,56	38,89	41,92	45,64	48,29
27	36,74	40,11	43,19	46,96	49,65
28	37,92	41,34	44,46	48,28	50,99
29	39,09	42,56	45,72	49,59	52,34
30	40,26	43,77	46,98	50,89	53,67
40	51,81	55,76	59,34	63,69	66,77
50	63,17	67,50	71,42	76,15	79,49
60	74,40	79,08	83,30	88,38	91,95
70	85,53	90,53	95,02	100,43	104,21
80	96,58	101,88	106,63	112,33	116,32
90	107,57	113,15	118,14	124,12	128,30
100	118,50	124,34	129,56	135,81	140,17

ANEXO 4

ENCUESTA DIRIGIDA A LOS DESARROLLADORES DE SOFTWARE DE LA ESCUELA DE SISTEMAS

OBJETIVO.- Determinación del conocimiento de metodología para la especificación de requisitos de software orientados a objetos y su aplicación.

1. ¿Conoce usted alguna metodología para Especificación de requisitos de software orientada a objetos?

   SI ☐ NO ☐

2. ¿Aplicó alguna metodología para la especificación de requisitos de software en su desarrollo de software?

   SI ☐ NO ☐

3. ¿En el desarrollo de software que usted realizó generó documentos de Especificación de requisitos de software antes de empezar a programar?

   SI ☐ NO ☐

4. ¿Ha utilizado Herramientas CASE para la Especificación de requisitos de Software?

   SI ☐ NO ☐

   En caso de Si

   ¿Qué herramienta utilizó? _____

5. ¿Si utilizó una herramienta CASE califique la experiencia de la utilización de esta herramienta?

   Excelente ☐ Muy Bueno ☐ Bueno ☐ Regular ☐ Malo ☐

6. Podría decir que el software que usted desarrollo se resume en:

   Mantener una base de datos (ingreso, actualizaciones, etc.) ☐

   Ejecutar funciones complejas para encontrar un resultado ☐

   Presentar gráficos, animaciones, sonidos, etc. (Multimedia) ☐

ENTREVISTA DIRIGIDA A LOS DIRECTORES DEPARTAMENTALES DE LAS ENTIDADES PARA LAS CUALES SE HA DESARROLLADO SOFTWARE

OBJETIVO.- Evaluación del software desarrollado en las prácticas pre - profesionales y tesis de grado para determinar si estos han satisfecho los requisitos de los usuarios finales

1. ¿Se entregó documentación junto con el sistema desarrollado?

   SI ☐ NO ☐

   ¿Cuál?

   Manual de Usuario ☐
   Manual Técnico ☐
   Especificación de Requisitos de Software ☐
   Otros …… ☐

2. ¿El sistema desarrollado para su departamento está siendo utilizado?

   SI ☐ NO ☐

   Si no, por qué?

3. Si sí….. ¿Las funciones que realiza el sistema son las que se deseaban antes de que se desarrollará?

   SI ☐ NO ☐

4. ¿En qué porcentaje están satisfechos con el funcionamiento del sistema?

   0% ☐ 25% ☐ 50% ☐ 75% ☐ 100% ☐