



UNIVERSIDAD AUTÓNOMA DE QUERÉTARO  
FACULTAD DE INFORMÁTICA  
MAESTRÍA EN SISTEMAS DE INFORMACIÓN

**Estudio, propuesta e implementación de un proceso mejorado para el levantamiento de requerimientos dentro de la fase de análisis de un proyecto de TI.**

**T E S I S**

Que como parte de los requisitos para obtener el grado de Maestría en Sistemas de Información

Presenta:


**I.S.C. Laura Clemencia Rodríguez Morales**

Dirigido por:

**M.C.C. Alejandro Madariaga Navarrete**

**SINODALES**

M.C.C. Alejandro Madariaga Navarrete  
Presidente

  
Firma

M.I.S.D. Juan Salvador Hernández Valerio  
Secretario

  
Firma

Dra. Marina González Herrera  
Vocal

  
Firma

Dr. Joaquín Agustín García Rodríguez  
Suplente

  
Firma

Mtra. Martha Yolanda Rodríguez Avilés  
Suplente

  
Firma



M.I.S.D. Juan Salvador Hernández Valerio  
Director de la Facultad de Informática



Dra. Ma. Guadalupe Flavia Loarcá Piña  
Directora de Investigación y Posgrado

## RESUMEN

La elicitación de requerimientos se refiere al proceso en que el cliente establece con claridad todos los elementos que necesita en su sistema (Zapata *et al*, 2010). Sin embargo las Pymes deben ofrecer desarrollo en tiempo récord y con el mínimo invertido en la documentación, omitiendo la elicitación de manera formal. Esta tesis establece que con un proceso simple de levantamiento diseñado para una Pyme es posible disminuir las modificaciones no pagadas por el cliente, los retrasos y mejorando la sensación de calidad en el cliente. Se diseñó un proceso de elicitación de requerimientos: PROSER, este se comparó con el proceso tradicional en un proyecto real realizado para una empresa educativa de gran prestigio ubicada en la ciudad de Querétaro. El proyecto se cubrió durante la separación de la empresa en preparatoria y universidad. Se dividió el proyecto en 38 módulos de los cuáles 19 y 19 se desarrollaron con los dos distintos procesos. Las variables sujetas a comparación fueron: número de modificaciones, días de retraso y aceptación del cliente. La tercera variable fue tomada según la opinión de los desarrolladores. En el tiempo de retraso se compararon los días estimados en ambos proyectos con los días que tardaron en terminarse en la realidad, la tasa de retraso fue con PROSER= 24%, mientras que la tasa de retraso tradicional = 54% . La media de los días de retraso del proceso Tradicional fue de  $\bar{x}=12$  y  $s=8$  mientras que con PROSER la media  $\bar{x}=1,9$  y la  $S=1.35$ . La media de modificaciones en el proceso tradicional, modificaciones iniciales, tradicional  $x=15$   $s=9$  Vs PROSER  $x=4$  y  $s=3$ , modificaciones finales fue de  $\bar{x}=6$  y  $s=2$  Vs PROSER  $\bar{x}=2$  y  $s=1.3$ . La media del porcentaje de aceptación en el proceso tradicional es de  $\bar{x}=80$ , mientras que con PROSER el promedio fue de  $\bar{x}=86$ . De manera general los resultados indican que el PROSER logró disminuir los días de retraso, explicándose por la mayor claridad que brindó el levantamiento de requerimientos inicial y contiene más retroalimentación antes de la entrega final, este proceso esclarece el levantamiento de requerimientos y permite que el proyecto se desarrolle y concluya con mejor relación con el cliente. A esta Pyme y otras con similares características les sería altamente conveniente invertir en este proceso.

**Palabras clave:** (Elicitación de requerimientos, Pyme, proceso de desarrollo de sistemas desarrolladores)

## SUMMARY

Requirements elicitation is the process by which the client establishes clearly all the elements needed in its system (Zapata et al., 2010). Nevertheless the SMBs must offer development in record time and with the minimum time invested in documentation, omitting the formal requirements elicitation. This thesis establishes that: with a simple process of requirements elicitation designed for a SMB it is possible to diminish unpaid modifications, delays and improve the perceived quality by the client. A process for requirements elicitation was designed: PROSER, which was compared with the traditional process in a real project for a renowned company in the Querétaro city. The project was carried out during the separation of the institution in High School and University. The project was divided into 38 modules of which half of them were developed with one process and the other half with the other. The variables subject to comparison were: number of modifications, days of delay and acceptance by the client. The third variable was measured based on the developer's opinion. For delay time was estimated duration in days minus real duration in days divided between the real duration in days for both portions of the project, the delay was with PROSER = 24 %, while the traditional delay was = 54 %. The average of the delay in days of the Traditional process was of  $\bar{x}=12$  and  $s=8$  while with PROSER the average  $\bar{w}$  was  $\bar{x}=1.9$  and the  $s=1.35$ . The modifications average for the traditional process, initial modifications, traditional  $\bar{x}=15$  and  $s=9$  vs. PROSER  $\bar{x}=4$  and  $s=3$ . Final modifications was  $\bar{x}=6$  and  $s=2$  for traditional vs. PROSER  $\bar{x}=2$  and  $s=1.3$ . The percentage average of acceptance in the traditional process was  $\bar{x}=80$ , while with PROSER the average was  $\bar{x}=86$ . In general these results show that PROSER reduced the delay in days. This is explained by the greater clarity in the initial requirements elicitation and the better feedback received before final delivery. This process clarifies the requirements elicitation and allows the project to develop and to conclude, in a better relationship with the client. To this SMB and others with similar characteristics it would be highly suitable to invest in implementing this process.

**Key words:** (System requirements, design of a system, small and medium-sized enterprise/businesses, developers)

“Tu trabajo va a llenar gran parte de tu vida, y la única forma de estar realmente satisfecho con él es hacer lo que creas que es un gran trabajo. Y la única manera de hacer un trabajo genial es amar lo que haces. Si no lo has encontrado, sigue buscando. No te detengas. Al igual que con todos los asuntos del corazón, lo sabrás cuando lo encuentres. Y, como cualquier gran relación, sólo se pondrá mejor y mejor, conforme los años pasen. Así que sigue buscando hasta que lo encuentres.

No te detengas.”

-Steve Jobs

“Yo creo que fuimos nacidos hijos de los días, porque cada día tiene una historia y nosotros somos las historias que vivimos...”

-Eduardo Galeano

## **AGRADECIMIENTOS**

A mí papá Don Felipe Rodríguez Guerrero.

Agradezco a mi familia Noé, Clara, Francisco y Felipe, por darle sabor a mi vida y ser mi razón para pensar en más.

Al Maestro Alejandro Madariaga Navarrete por su apoyo y su confianza.

# ÍNDICE

I. INTRODUCCIÓN.....	1
Planteamiento del problema .....	3
Hipótesis.....	4
Objetivos .....	4
Objetivo general .....	4
Objetivos particulares.....	4
Justificación.....	5
Alcances .....	6
<b>Delimitaciones</b> .....	7
II. REVISION DE LITERATURA.....	8
Ingeniería de software .....	10
Modelo de ciclo de vida clásico .....	10
Modelo RAD.....	13
Modelo de prototipos.....	14
Enfoque de desarrollo de software ágil (Development agile) .....	16
SCRUM.....	18
Elicitación de requerimientos .....	20
Técnicas de elicitación de requerimientos.....	21
Características óptimas de un requerimiento.....	22
Retos que se presentan en la elicitación de requerimientos.....	23
Requerimientos de calidad .....	24
Software de calidad .....	25
Satisfacción del cliente.....	26
Tiempo de entrega .....	27
Errores en los requerimientos.....	28
Software comercial para la gestión de proyectos .....	29
Las Pyme en México .....	29
III. METODOLOGÍA .....	32
Objeto de estudio.....	32
Variables.....	34
Proceso de elicitación de requerimientos tradicional en TImobile.....	36

Diseño y comparación con el proceso PROSER.....	39
IV.    RESULTADOS .....	41
Diseño del proceso simple para la elicitación de requerimientos (PROSER).....	41
Creación de la plantilla SRS de PROSER.....	46
Tiempo de retraso: Tradicional Vs PROSER.....	48
Días de retraso Vs días totales.....	51
Modificaciones proceso Tradicional Vs Modificaciones PROSER.....	53
Porcentaje de aceptación Tradicional Vs PROSER .....	58
V.    DISCUSIÓN.....	62
VI.    BIBLIOGRAFÍA.....	65

## ÍNDICE DE FIGURAS

### Figura

1	Ciclo de vida clásico del software .....	11
2	Modelo RAD .....	14
3	Modelo de prototipos.....	15
4	Agilidad y el costo del cambio. ....	17
5	Proceso Scrum.....	18
6	Proceso de elicitación de requerimientos Tradicional en TImobile. ....	38
7	Proceso Simple de Elicitación de Requerimientos PROSER en contacto con el cliente. ....	44
8	PROSER etapa de documentación para el manual.....	46

## ÍNDICE DE TABLAS

### Tabla

1	Módulos proceso tradicional .....	48
2	Módulos proceso PROSER .....	48
3	Días de retraso proceso tradicional .....	49
4	PROSER, días estimados Vs días retraso.....	50
5	Tradicional, días estimados Vs días totales.....	51
6	PROSER, días estimados Vs días totales .....	51
7	Modificaciones proceso tradicional.....	54
8	Modificaciones PROSER.....	54
9	Organización de módulos para su comparación.....	56
10	Porcentaje de aceptación proceso tradicional.....	59
11	Porcentaje de aceptación PROSER .....	59

## ÍNDICE DE GRÁFICAS

### Gráfica

1	Tradicional, días estimados Vs días de retraso.....	49
2	PROSER, días estimados Vs días de retraso .....	50
3	PROSER, días estimados Vs días totales .....	52
4	Tradicional, días estimados Vs días totales.....	52
5	Media días de retraso.....	53
6	Modificaciones proceso tradicional.....	55
7	Modificaciones PROSER .....	55
8	Modificaciones iniciales tradicional Vs PROSER .....	57
9	Modificaciones finales proceso tradicional Vs PROSER.....	58
10	Aceptación tradicional Vs PROSER .....	60
11	Medias de porcentaje de aceptación .....	61





# I. INTRODUCCIÓN

La obtención de requisitos es cuando los interesados en un sistema de software descubren, entienden y expresan organizadamente sus requisitos, en numerosas ocasiones requiere de tiempo lograr especificar con claridad lo que el interesado espera del software, por eso es necesario de parte de los analistas o desarrolladores conocer técnicas que permitan comunicación adecuada con el cliente para establecer con claridad los requisitos y lograr su satisfacción (Raghavan *et al*, 1994).

Se le ha denominado obtención de requisitos, levantamiento de requerimientos, elicitación de requerimientos y educación de requisitos, en general se refieren a que el cliente establezca y exprese con claridad todos los elementos que necesita en su sistema, sin embargo esto representa una enorme dificultad, pues muchas ocasiones requieren que el analista investigue qué es lo que el cliente busca haciendo uso de técnicas especiales para ello (Zapata *et al*, 2010).

Es por esto que el levantamiento o elicitación de requerimientos se convierte en un trabajo especializado más allá de sólo levantar una lista de requisitos, incluso, en el término educación de requisitos, la educación se refiere a “*sacar una cosa de otra*” haciendo referencia a las complicaciones que tiene esta labor que pone al analista un trabajo mucho más activo que el de un receptor de información.

La elicitación de requerimientos es la parte del proceso sobre la cual se sostiene todo proyecto en desarrollo, es la base de las etapas posteriores del ciclo de vida del desarrollo de software pues de ello dependerá la elaboración del modelo conceptual del sistema (Young, 2006). En esta fase los interesados descubren, articulan y entienden sus requisitos, el analista organiza y detalla los requisitos de los interesados, realiza evaluaciones para evitar conflictos e inconsistencias e identifica los requisitos faltantes (Zapata *et al*, 2010).

En el trabajo cotidiano del desarrollador esta fase es crucial que se haga explícitamente ordenada o no, sin embargo si se realiza adecuadamente tiene un alto impacto en las otras fases disminuyendo los cambios y correcciones en los requisitos ayudando así al máximo aprovechamiento de recursos temporales y humanos dentro de un proyecto.

Ahora bien, la calidad de la elicitación debe establecer con exactitud la retroalimentación al cliente acerca de los requisitos planteados, así que requiere cercanía a la perfección, ajustarse a las necesidades del cliente, al tamaño del proyecto e incluso a la compañía desarrolladora de software (Manies y Nikual, 2011) y debe iniciarse inmediatamente después de la planeación de negocios con el cliente (Goguen y Linde, 1993).

Así, la elicitación de requerimientos se convierte en una fase crítica para la satisfacción del cliente, para el buen desarrollo del sistema para el analista y el buen uso de recursos para la empresa de desarrollo. Gracias a la experiencia como analista de software en una empresa radicada en Querétaro es posible generar este trabajo de investigación que se propone como tesis de Maestría para obtener el grado de Maestra en Informática: conjugando los conocimientos obtenidos en los cursos, la literatura y la experiencia de más de 10 años en el área.

TImobile es una empresa radicada en Querétaro dedicada al desarrollo de software, actualmente se enfrenta a los mismos retos de otras empresas de tecnología en México, y de no tecnología, tales como la obtención de clientes, la búsqueda de la satisfacción de los clientes y la optimización de recursos. Adicional a ello, realizando un análisis de las áreas de oportunidad de la empresa en la búsqueda de mejorar.

Durante la búsqueda de áreas de oportunidad de la empresa se observó que de manera recurrente se encuentra la empresa en situaciones de retraso en los tiempos de entrega, se analizaron posibles causas de este problema y una de las posibles causas es la falta de un proceso adecuado de elicitación y documentación de requerimientos para las necesidades y tamaño de esta empresa, si bien es sólo posible que este sea el motivo de retraso, resulta evidente, como lo señala la teoría, que una mejora en esta fase puede transformarse en una mejora en los tiempos de entrega.

Por esto, se pretende del producto de este trabajo de investigación, un protocolo de levantamiento de requerimientos que se pueda aplicar en esta y cualquier empresa pequeña o mediana de desarrollo de software, que comparta ciertas características, que deseé mejorar su proceso de elicitación de requerimientos y que gracias a su uso, disminuir la cantidad de modificaciones al producto y obtener mayor satisfacción de los clientes.

## **Planteamiento del problema**

Esta tesis se fundamenta en dos realidades, los clientes que solicitan un software a la medida con frecuencia no tienen absoluta claridad en lo que desean del software y los desarrolladores de software, también con frecuencia no desarrollan lo que el cliente pide, sino lo que ellos han comprendido de los requerimientos del software.

Esta confusión genera gastos en tiempo de re programación, cambios no pagados por el cliente, pues se argumenta que se había pedido, frustración en el desarrollador y la sensación de falta de calidad en el producto para con el cliente.

Actualmente existe una amplia gama de literatura en torno al levantamiento de requerimientos en grandes empresas, sin embargo, la documentación suele ser tan engorrosa que se pasa por alto en pequeñas empresas que requieren optimizar el tiempo de trabajo de sus empleados para continuar en la competencia del mercado.

De esta manera se propone desarrollar un proceso que permita ahorrar tiempo en re programación, disminuyendo las modificaciones al código, que permita al desarrollador conocer lo que el cliente desea de manera objetiva, ayude a generar un claro consenso entre lo que el cliente pide y lo que el desarrollador hace, separando los cambios generados por más potencialidades del software y los errores del desarrollador.

Una plantilla simple que mejore la sensación de calidad en el cliente, es decir tenga un efecto positivo en la satisfacción del cliente.

Y finalmente, se ofrezca como una oportunidad de mejora a las Pymes con problemas similares.

## **Hipótesis**

¿Podrá un proceso de levantamiento de requerimientos diseñado a la medida para una Pyme de desarrollo de software, disminuir el tiempo de reprogramación, disminuir la cantidad de modificaciones al código, mejorando la claridad entre los deseos del cliente y lo comprendido por el desarrollador, ofreciendo mayor satisfacción al cliente y al desarrollador?

## **Objetivos**

### **Objetivo general**

Diseñar y probar un proceso ordenado y simple de elicitación de requerimientos, que se adapte a las necesidades reales de una Pyme desarrolladora de software. Un proceso que reduzca las modificaciones al código, disminuyendo los retrasos en la entrega, permita aclarar y verificar los deseos del cliente reduciendo las disparidades entre lo que pide el cliente y lo que se entrega, mejorando así, la satisfacción del cliente.

### **Objetivos particulares**

1. Diseñar un proceso que se ajuste a las necesidades de una Pyme sin lo engorroso de los ya establecidos para grandes corporaciones.
2. Diseñar una plantilla resumida en la que se describan con claridad las necesidades del cliente.
3. Probar este nuevo proceso en un proyecto real.
4. Comparar en el proceso tradicional y el nuevo proceso, aplicando ambos en un mismo proyecto en diferentes fases.
5. Comparar en ambos procesos el tiempo de retraso.
6. Comparar la cantidad de modificaciones solicitadas por el cliente.
7. Comparar el porcentaje de aceptación, interpretado como satisfacción del cliente.
8. Conocer las opiniones del equipo de trabajo ante la posibilidad de implementar este proceso.

## **Justificación**

El levantamiento de requerimientos es una etapa esencial en el arranque de todo proyecto de desarrollo de software y debe realizarse efectivamente para poder aumentar las garantías de éxito en los proyectos (Alarcón, 1996).

Muchos profesionistas no realizan correctamente esta fase porque en sus empresas no hay procesos que los apoyen para realizarlas, en este mismo sentido, no hay procesos debido a que los que existen pueden ser engorrosos y requieren de invertir un período de tiempo largo que a simple vista puede parecer una “pérdida de tiempo”.

Ahora bien un software debe solucionar una necesidad de una empresa, existen en un software a la medida generalmente, varias personas de la empresa involucradas en el proyecto, los desarrolladores, y todos deben involucrarse en el desarrollo del sistema, conociendo a profundidad los procesos del cliente. Para conocer los procesos del cliente y llevar a cabo un exitoso software que le brinde facilidades y comodidad al cliente es imprescindible que el desarrollador conozca los deseos del cliente y prepare una serie de pasos ordenados para su correcto desarrollo (Torres y Noda, 2012).

No siempre esto ocurre en el inicio de un proyecto, debido a que se puede considerar que se ahorra tiempo en el levantamiento formal.

Sin embargo de manera recurrente el cliente argumenta que eso no fue lo que quería decir, hace cambios que no paga, generando cambios importantes al código, pérdida de tiempo, retrasos en la entrega, sensación de frustración en el desarrollador y finalmente la sensación de falta de calidad en el cliente.

Sin embargo, un proceso ordenado de levantamiento de requerimientos puede llevarnos a invertir más tiempo al inicio, fomentar la calidad en el sistema, mejorar la sensación de hacer bien las cosas en el desarrollador, disminuir los tiempos de reprogramación, pues los cambios del cliente, deberán ser pagados, pues surgirán como potencialidades del sistema, no como errores del programador (Zapata, Giraldo y Mesa, 2010).

Por todo esto la importancia de diseñar un proceso en las Pymes, que les permita ahorrar tiempo de reprogramación, pero que no implique un engorroso proceso diseñado para grandes corporaciones.

Esto es lo que se plantea este trabajo de tesis, diseñar un proceso a la medida de una pyme que permita mejorar aquellos aspectos relacionados con la ausencia de un proceso y que a la vez logre tener los beneficios de un proceso ordenado, entregas a tiempo, control sobre los requerimientos, menos modificaciones al código, claridad en el análisis y control sobre lo que el cliente pide y lo que se desarrolla, mejorando por ende la sensación de calidad en el cliente.

## **Alcances**

Mediante la literatura existente sobre el levantamiento de requerimientos y la experiencia de 10 años de trabajo como desarrollador en la empresa, se diseñará un proceso a la medida de una Pyme ubicada en la ciudad de Querétaro: TImobile.

En este trabajo de tesis se recorrerá un poco sobre la historia de la computación como disciplina, se resumirán algunos de los procesos de levantamiento de requerimientos más populares alrededor del mundo, se revisarán sus virtudes y dificultades.

Se diseñará el proceso de forma que permita tener claridad entre el cliente y el desarrollador, disminuyendo así el tiempo de re programación.

Una vez diseñado el proceso se probará en un proyecto real, asimismo se realizarán algunas comparaciones con el proceso anterior que será comparado a través del mismo proyecto pero en diferentes módulos, se compararán las variables de número de modificaciones al código, el tiempo de re programación y la satisfacción del cliente.

## **Delimitaciones**

Con respecto a la implementación definitiva del proceso, no se tiene como objetivo implementarlo en toda la empresa TImobile como un proceso a seguir, si bien se plantea que sea usable y tenga beneficios a los desarrolladores, a la empresa y a los clientes, también se reconoce que ese esfuerzo corresponde a la empresa particular y no a los alcances de este proyecto. Así que se limitará a probarse y evaluarse en los términos dictados en la metodología.



## II. REVISION DE LITERATURA

Durante el desarrollo de la computación como disciplina formal, existieron periodos de relevante importancia, entre ellas, el surgimiento de las principales tareas especializadas dedicadas a los analistas o desarrolladores, el surgimiento y evolución del software y hardware así como la estandarización de procesos, a continuación se hace un breve recorrido sobre dichas etapas.

En el período de los inicios de la computación, surgieron varias computadoras, entre ellas la ENIAC (Electronic Numerical Integrator And Computer) por sus siglas en inglés, que fue construida en la Universidad de Pensilvania por John Presper Eckert y John William Mauchly (Ramos, 1998).

La computadora ENIAC requería la operación manual de cerca de seis mil interruptores y su programa o software cuando requería modificaciones, demoraba semanas de instalación manual, un dato interesante es que esta computadora fue programada por seis mujeres, quienes sentaron las bases para que la programación fuera sencilla y accesible para todos, estas seis mujeres crearon el primer set de rutinas, las primeras aplicaciones de software y las primeras clases en programación, su trabajo fue muy relevante para la evolución de la programación entre las décadas de los 40 y 50 (Kleinman, 2001).

Para los años 50 el desarrollo de software se comenzó a realizar con el mismo proceso que el desarrollo del hardware, el proceso de tipo cascada. En los años 60 el desarrollo de software era artesanal, en las propiedades de software se manejaban los concepto de: fáciles de modificar, fácil de copiar, no se gasta y es invisible, en esta época se fomentó el proceso de desarrollo tipo “codifica y corrige”, en esta misma década se inició la cultura del hacker en el buen sentido de la palabra, es decir experto en programación, y la del “cowboy” que hace desarrollos heroicos de última hora (Oktaba, 2008).

En 1964 el sistema 360 de IBM combinó por primera vez las características científicas y de negocio lo que encaminó a un numeroso grupo de personas a tratar de desarrollar software para grandes sistemas físicos y de gestión, en 1969 Bauer acuñó el término “Ingeniería de Software” (Mohapatra, 2010).

A finales de los años 60 ocurrió una crisis importante en el uso del software debido a la implementación de los circuitos integrados al hardware, que hizo posible lo que no lo era mediante el software, comenzaron así, los proyectos grandes y complejos, en los que existían retrasos y sobrecostos, todo esto por falta de metodologías establecidas que permitieran un adecuado proceso en el desarrollo del software, entre los métodos faltantes se encontraba la especificación de requerimientos, implementación y diseño (Sommerville, 2005).

Las fases del desarrollo que actualmente siguen en uso fueron identificadas y clasificadas en la década de los 70, requerimientos, análisis, diseño, codificación y pruebas, se introdujo además en esta década la programación estructurada y los métodos formales para especificar el software, además se identificaron los principios de diseño, como modularidad, encapsulación, abstracción de tipos de datos, acoplamiento débil y alta cohesión, entre otros. En esta misma década se publica el modelo de cascada y se definen los conceptos de verificación y validación (Mohapatra, 2010).

En los años 80 se busca la productividad y escalabilidad de sistemas y equipos de desarrollo. Renace con fuerza la programación orientada a objetos a través de las múltiples propuestas de lenguajes de programación. Se crea el primer modelo de madurez de capacidades de procesos (SW-CMM) y los primeros estándares. Nace el concepto de Fábricas de Software y se generan las primeras herramientas para incrementar la productividad a través de la programación por el usuario, tales como los lenguajes de cuarta generación denominados 4GLs (Ramos, 1998).

En los años 90 la concurrencia adquiere mayor importancia con respecto a procesos secuenciales, la orientación a Objetos se extiende a las fases del análisis y diseño, se acuerda un lenguaje de modelado denominado UML y se genera el primer proceso comercial de desarrollo orientado a objetos RUP. Los diseñadores y los arquitectos de software empiezan a recaudar las mejores experiencias a través de patrones de diseño y de arquitectura. También en esa década se define el Modelo Espiral para el desarrollo basado en el análisis de riesgos y su vertiente conocida como desarrollo iterativo e incremental. Uno de los mayores cambios de la década fueron los cambios generados por el software libre, el cual tomó gran fuerza y dio paso a la creación de los primeros ejemplos exitosos, también la usabilidad de sistemas se convirtió en el foco de atención e investigación, asimismo el software mediante el desarrollo web comenzó a ocupar un lugar fundamental en el mercado competitivo y en la sociedad (Oktaba, 2008).

## **Ingeniería de software**

La ingeniería de software es una disciplina de la ingeniería que involucra todo el proceso del software desde la especificación del sistema hasta el mantenimiento cuando el sistema ya está en producción. Como disciplina que parte de la ingeniería debe tratar de encontrar soluciones utilizando teorías, métodos y herramientas para hacer que las cosas funcionen.

Por otra parte, su ejercicio debe comprender el todo y se le denomina: “proceso de vida de software” y comprende no solo el desarrollo, también la gestión de software, desarrollo de herramientas, métodos y teorías que apoyan a la producción del software (Pressman, 1997).

Un proceso de software son todas las actividades relacionadas que llevan a la producción de un producto de software, existen modelos que plantean los procedimientos para llevar un proceso de software de calidad y por lo tanto tener como resultado un software de calidad (Sommerville, 2005).

Así, la ingeniería de software es una consecuencia del hardware y la ingeniería de sistemas que abarca en conjunto los tres elementos clave, los métodos, herramientas y procedimientos que permiten al administrador controlar el proceso de desarrollo de software.

Los métodos aportan la técnica para la construcción del software, es decir, el *cómo*; las herramientas aportan el soporte automatizado o semi automatizado para los métodos; y los procedimientos definen la secuencia de aplicación de los métodos, los entregables, los controles y los hitos (Pressman, 1992).

### **Modelo de ciclo de vida clásico**

En el ciclo de vida del software, la primera etapa se denomina análisis previo y también análisis de sistemas o ingeniería de sistemas, en esta etapa se definen a grandes rasgos los aspectos generales del software, la segunda etapa es el análisis de requisitos o simplemente análisis, su objetivo es definir con detalle las necesidades de información que tendrá que resolver el software,

sin tener en cuenta los medios técnicos con los que se tendrá que llevar a término el desarrollo de software (Campderrich, 2003).

La figura 1 muestra el ciclo de vida clásico del software según Campderrich (2003) en él, es posible apreciar que el primer paso es el análisis previo, para saber qué es lo que necesita hacer el software, en segundo lugar es la etapa de análisis de requisitos en donde se debe especificar y documentar claramente lo que el software deberá realizar, enseguida lugar vienen las etapas de diseño, programación, prueba y mantenimiento.

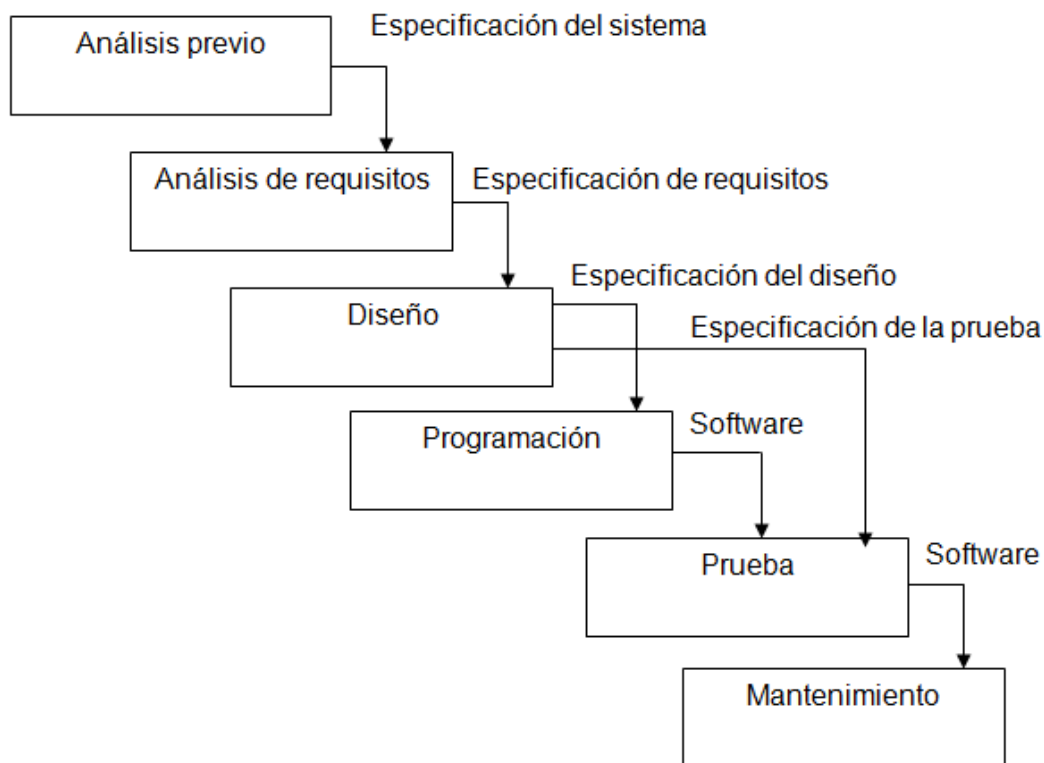


Figura 1 Ciclo de vida clásico del software  
Fuente: Campderrich, 2003.

Este ciclo de vida clásico del software se puede dividir en tres fases:

1. Planificación: contiene las etapas de análisis previo y análisis de requisitos
2. Desarrollo: con las etapas de diseño, programación y prueba

### 3. Mantenimiento

Dentro de la fase de planificación se encuentra el análisis del sistema el cuál se resume en un documento llamado *Especificación del sistema*, que servirá como base para la ingeniería del hardware, del software, la base de datos y la interacción con los humanos, Alonso y colaboradores (2005) proponen un índice de contenido para documentarlo adecuadamente:

#### a) Objetivos generales del sistema

- ⇒ Fines del sistema
- ⇒ Funcionamiento y rendimiento requerido
- ⇒ Entradas al sistema
- ⇒ Restricciones

#### b) Definición de requisitos del:

- ⇒ Software
- ⇒ Hardware
- ⇒ Bases de datos
- ⇒ Personal y otros elementos del sistema

#### c) Análisis técnico

- ⇒ Rendimiento técnico
- ⇒ Fiabilidad
- ⇒ Mantenimiento
- ⇒ Producción

#### d) Análisis económico

- ⇒ Costos de producción
- ⇒ Costos de mantenimiento
- ⇒ Relación coste/beneficio

#### e) Viabilidad del sistema

- ⇒ Económica y de mercado
- ⇒ Técnica
- ⇒ Legal
- ⇒ Bibliografía y apéndices, se presentan las tablas de datos, la descripción detallada de los algoritmos, los diagramas, los gráficos y demás material de interés.

Sin embargo un proyecto real raramente sigue el flujo secuencial de este modelo así que es posible encontrar iteración indirecta (Pressman, 2005).

## **Modelo RAD**

El modelo de desarrollo rápido de aplicaciones o Rapid Application Development (RAD), es un proceso de desarrollo de software que enfatiza un ciclo de desarrollo corto, mantiene un desarrollo iterativo y es recomendable en la construcción de prototipos, este modelo permite al equipo de desarrollo crear un sistema totalmente funcional en poco tiempo, si los requerimientos son adecuadamente entendidos, cuando el alcance del proyecto es limitado (Mohapatra, 2010).

En el modelo RAD la comunicación del equipo de trabajo es importante para entender los problemas del negocio y las características del sistema, la planeación es esencial porque múltiples equipos de trabajo pueden estar trabajando en paralelo en distintas funciones del sistema. El modelado comprende tres fases principales: modelado del negocio, modelado de datos y modelado del proceso, además establece las representaciones de diseño que sirven como base para la actividad de construcción en el modelo RAD (Pressman, 2005).

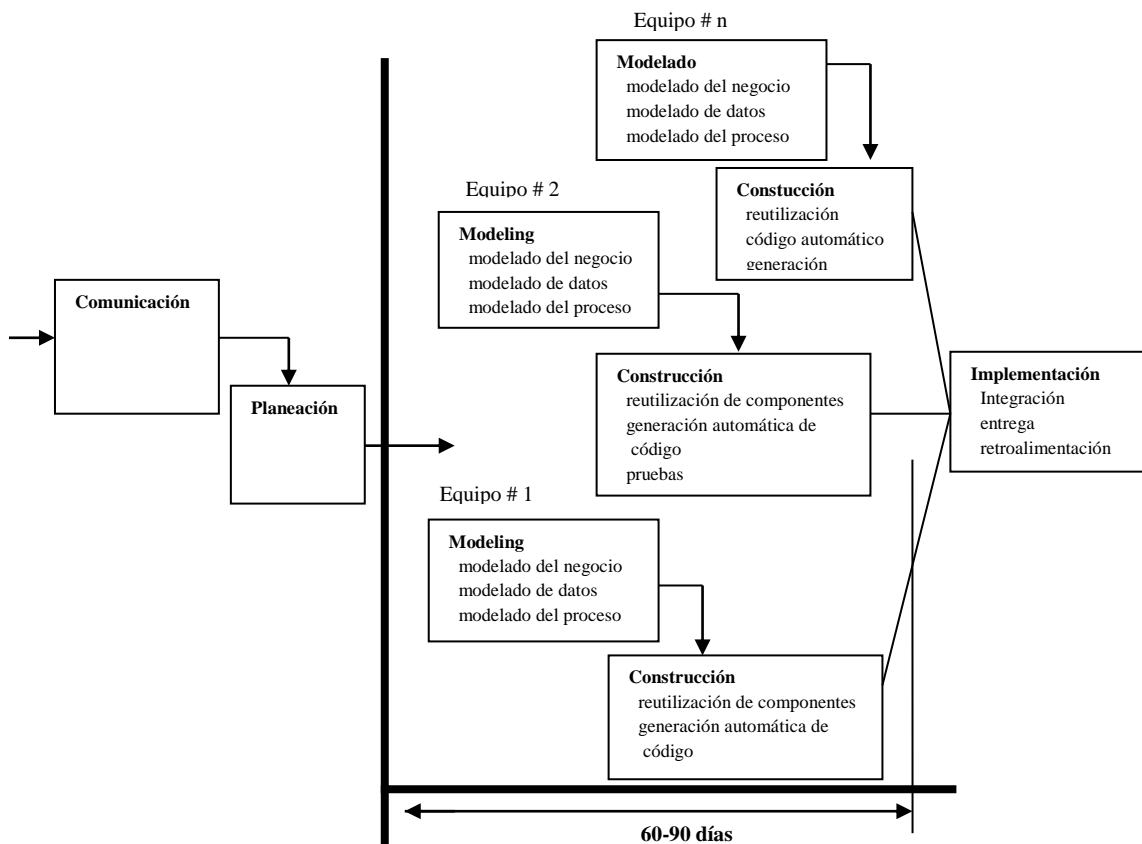


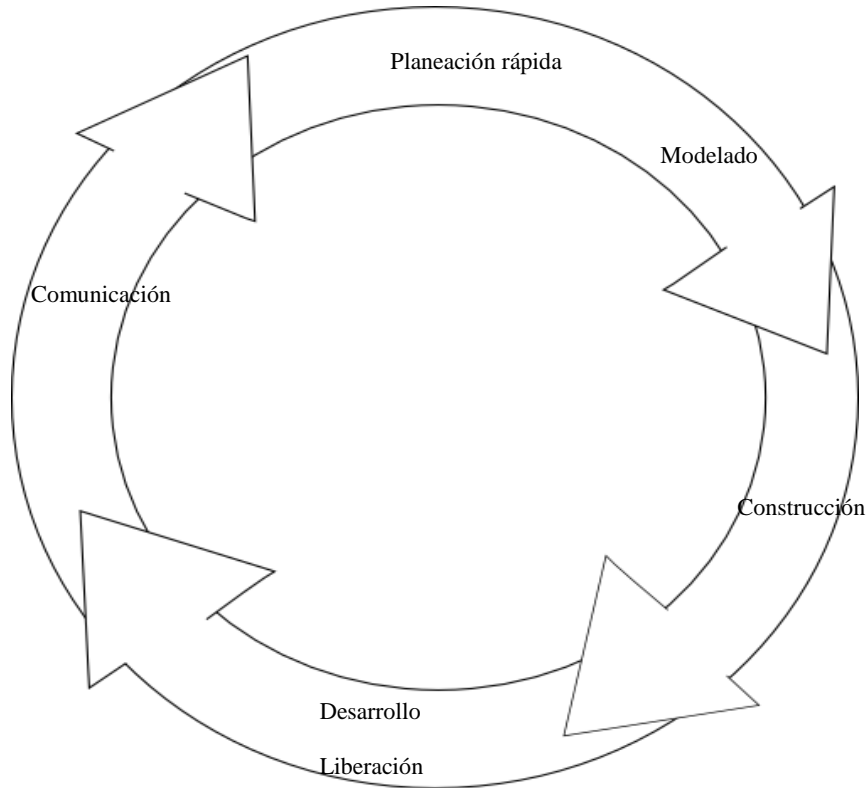
Figura 2 Modelo RAD  
Fuente: Pressman, 2005.

## Modelo de prototipos

En numerosas ocasiones el cliente tiene claros los objetivos generales del software que necesita pero es incapaz de detallar cuales serían los datos de entrada, el procesamiento que se necesita o los requerimientos de salida, en otros casos al desarrollador puede no estar seguro de la eficiencia del algoritmo que está realizando, la adaptabilidad de un sistema operativo o la manera en que el usuario y la computadora interactúan, cuando cualquiera de estos dos casos ocurre el paradigma de prototipos es la mejor opción.

El modelo de prototipos comienza con la comunicación, el desarrollador y el cliente se reúnen y definen todos los objetivos del software, identifican los requerimientos conocidos y es obligatorio realizar una especificación más precisa, en el modelo de prototipos ocurre la planeación

y el modelado rápidamente, el diseño rápido se enfoca en la representación de los aspectos del software que serán visibles para el cliente o usuario final, el prototipo es desarrollado y posteriormente evaluado por el cliente o usuario final, la retroalimentación es utilizada para afinar los requerimientos del software, la iteración en este modelo facilita la satisfacción de las necesidades del cliente y al mismo tiempo permite al desarrollador entender mucho mejor lo que se necesita hacer.



**Figura 3. Modelo de prototipos.**  
**Fuente: Pressman, 2005.**



## Enfoque de desarrollo de software ágil (Development agile)

Según Pressman (2005) el llamado movimiento del desarrollo de software ágil, se formó desde hace 2 décadas con el esfuerzo por superar las debilidades reales del desarrollo de software convencional.

El desarrollo ágil puede tener grandes ventajas pero no se puede aplicar a todos los proyectos, todos los productos, a toda la gente o a todas las situaciones, tampoco es la antítesis de las prácticas de desarrollo de software sólido y se puede aplicar en todos los trabajos de software como una filosofía base (Pressman, 1997).

Debido a que la economía actual tiene cambios constantes, es casi imposible predecir como evolucionara un proyecto de software a través del tiempo; las condiciones de mercado cambian rápidamente, los usuarios finales necesitan evolucionar y nuevas amenazas competitivas aparecen sin previo aviso, además existen situaciones en las que no se pueden tener los requerimientos totalmente definidos, lo que implica cambios y costos, el enfoque ágil es una alternativa eficaz ante estos factores pues logra reducir los costos de cambios a través del proceso de desarrollo (Páez *et al*, 2014).

Pero más que una respuesta eficaz al cambio, la agilidad incentiva una estructura en el equipo de trabajo y actitudes que hacen más fácil la comunicación entre los miembros del equipo, entre tecnólogos y empresarios, entre ingenieros de software operativo y los administradores, aunque, en general se presta atención en la entrega de software operacional y se deja un poco de lado la importancia de los productos intermedios; se adopta al cliente como parte del equipo de desarrollo y se trabaja en eliminar el *nosotros y ellos* actitud que prevalece en muchos proyectos de software y se reconoce que la planificación en un mundo incierto tiene sus límites y que un plan de proyecto debe ser flexible.

Se puede aplicar la agilidad a cualquier proceso de software, sin embargo, para lograrlo es esencial que el proceso se diseñe en términos que le permitan al equipo adaptar tareas y organizarlas, llevar a cabo la planificación de manera que entienda la fluidez de un enfoque de desarrollo ágil, eliminar todo pero mantener los productos más esenciales, e insistir en la estrategia

de la entrega incremental que genera software para el cliente tan rápido como es posible por tipo de producto y ambiente operacional (Pressman, 2005).

En cuanto a los costos de los cambios, Pressman (2005) pone en relieve que se sabe que incrementan de manera no lineal conforme progresa un proyecto, así, en la figura 4 la línea negra sólida muestra como incrementa la curva de costo del proyecto cuando el equipo está en medio de las pruebas de validación y el cliente solicita un cambio funcional, este cambio requiere una modificación en el diseño arquitectónico del software, el diseño y la construcción, implica nuevos diseños de pruebas, otros nuevos componentes y así sucesivamente, por lo que los costos aumentan rápidamente, el costo y el tiempo requeridos para asegurar que el cambio se realice sin efectos secundarios no deseados no es trivial.

Los defensores de la agilidad argumentan que un proceso ágil bien diseñado aplana la curva, con la línea gris sólida, lo que permite al equipo de desarrollo adaptarse a los cambios finales de un proyecto de software sin impactos importantes en el costo y el tiempo. Por ello, ya que el enfoque ágil incluye entregas incrementales junto con otras prácticas ágiles de pruebas continuas, el costo de hacer un cambio es atenuado (Pressman, 2005).

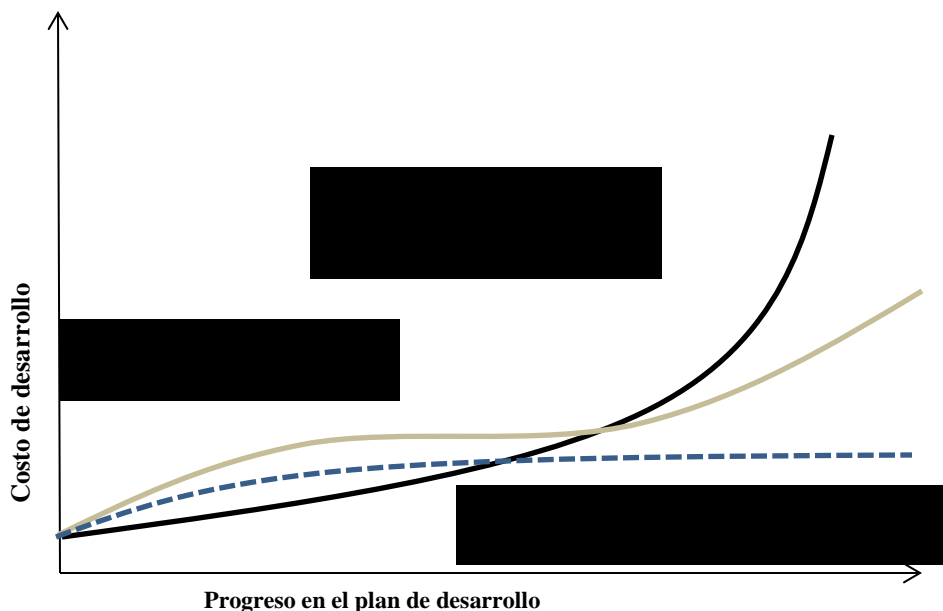


Figura 4. Agilidad y el costo del cambio.  
Fuente: Pressman, 2010.

## SCRUM

Es un método de desarrollo de software ágil concebido por Jeff Sutherland y su equipo de desarrollo a principios de 1990. En los últimos años el método de desarrollo Scrum ha sido transformado por Schwaber y Beedle.

El método de desarrollo Scrum es consistente con los principios del desarrollo ágil y son usados como guía para las actividades de desarrollo como son: levantamiento de requerimientos, análisis, diseño, evolución y entrega. En Scrum al conjunto de tareas que ocurren en un proceso se le llama *sprint* y el número de sprints requeridos en cada proyecto puede variar dependiendo de su complejidad y su tamaño (Schwaber y Beedle, 2001), el proceso completo del método Scrum se muestra en la figura 5.

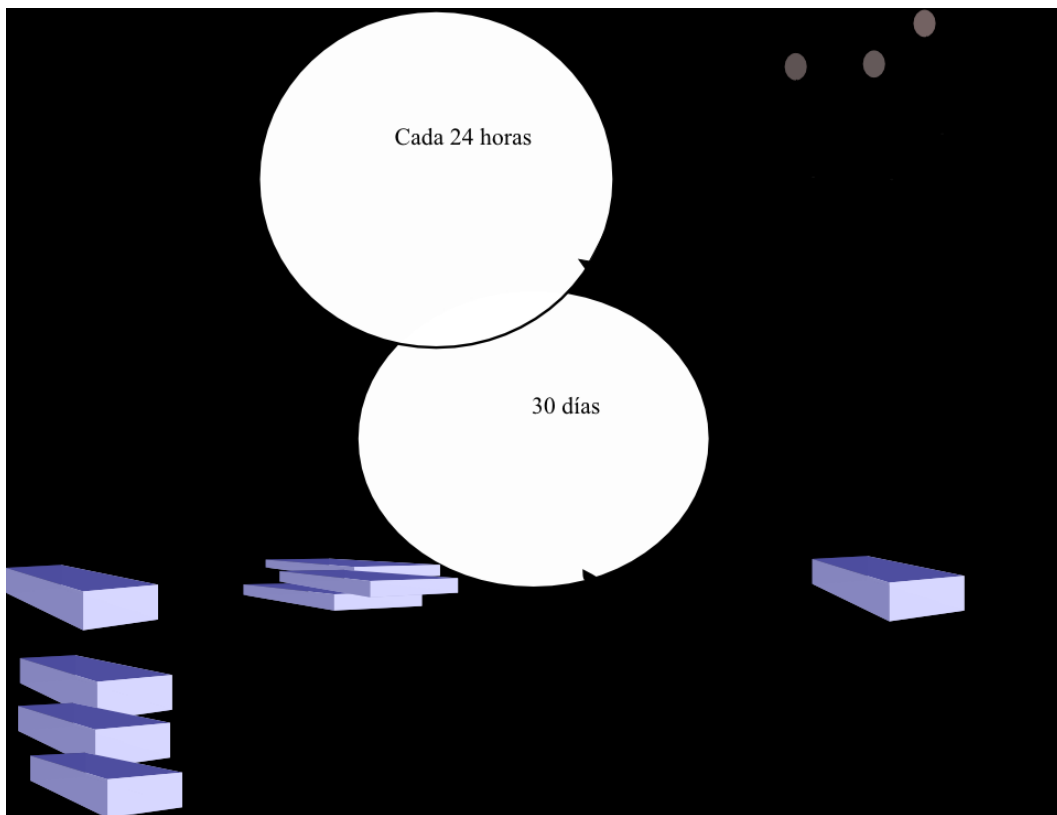


Figura 5. Proceso Scrum.  
Fuente: Pressman, 2005.

Según Schwaber y Beedle (2001) para utilizar Scrum se hace énfasis en la importancia que tiene el uso del conjunto de patrones de proceso de software que han demostrado efectividad para los proyectos con un tiempo corto, cada uno de estos patrones de proceso definen un conjunto de acciones de desarrollo:

- ⇒ Backlog, es una lista de los requerimientos priorizados o características que son valiosas para el cliente, pueden ser agregados elementos a esta lista en cualquier momento, lo que significa que se pueden introducir cambios, el administrador del proyecto evalúa el backlog y actualiza prioridades según se requiera.
- ⇒ Sprint, consiste en unidades de trabajo que son requeridas para lograr un requerimiento definido en el backlog dentro del periodo de tiempo indicado para el proyecto que generalmente es de 30 días, los cambios no se agregan durante el sprint, lo que permite al equipo trabajar en un tiempo corto y ambiente de desarrollo estable.
- ⇒ Scrum meetings, son reuniones diarias y cortas, generalmente de 15 minutos en las que se reúne e equipo de trabajo y se hacen tres preguntas principalmente:
  - ¿Qué se hizo desde la última reunión?
  - ¿Qué obstáculos se han encontrado?
  - ¿Cuál es el plan a lograr para la siguiente reunión?

Las respuestas son evaluadas por el líder del equipo, las reuniones Scrum sirven para descubrir problemas los problemas potenciales lo más pronto posible.

- ⇒ Demos, son liberables que pueden dar al cliente una visión más amplia de lo que se está desarrollando, el demo no precisamente contiene todas las funcionalidades solicitadas, las funcionalidades serán completadas durante el tiempo de desarrollo.

Ahora bien, hasta aquí con los modelos de desarrollo a continuación en esta segunda parte de la fundamentación teórica se hará énfasis en la elicitación de los requerimientos tomando como base lo que se ha descrito en la primer parte de esta fundamentación, donde se ha señalado a la especificación de requerimientos como la etapa base sobre la que se sostiene todo proyecto de desarrollo de software.

## Elicitación de requerimientos

Un requerimiento es una condición o capacidad que un sistema o componente de sistema debe lograr o poseer para así satisfacer un estándar, contrato, especificación u otro documento formalmente impuesto (Sommerville, 2005).

Los requerimientos son importantes debido a que proporcionan las bases para todo el trabajo posterior de desarrollo que se realiza, por lo que las bases deben ser lo más certeras que sea posible para tener los resultados deseados en el proyecto (Preece *et al*, 1994).

Existen dos tipos de requerimientos de software, los funcionales que son cualquier requerimiento que involucre una funcionalidad en el sistema, es decir, que defina específicamente como interactúa el sistema con el mundo exterior, el otro tipo es los requerimientos no funcionales, que involucran los aspectos de calidad en el software, como pueden ser el número de usuarios que trabajara con el sistema o las tecnologías con las que deberá ser compatible el sistema.

La elicitación de requerimientos es el proceso mediante el cual se documentan las necesidades que describen los usuarios de un sistema de software. La elicitación de los requerimientos debe documentarse adecuadamente de tal forma que la consulta y modificación de los mismos sea sencilla de realizar.

El principal objetivo de la elicitación de requerimientos es obtener un documento con las especificaciones precisas que nos dirán lo que el cliente desea y permitirá al equipo de desarrollo crear el producto correcto.

Las cifras nos dicen que el 53% de los proyectos de desarrollo técnico es víctima de los excesos de costos y proyectos fallidos (Sommerville, 2005).

Otra cifra alarmante es que entre en 40% y el 60% de los errores y defectos del software son el resultado de una pobre gestión y definición de requisitos (Manies y Uolevis, 2011).

Kevin Forsberg y Harold Mooz estudiantes de proyectos de la NASA encontraron que cuando la NASA dedicaba el 5% o menos del total del presupuesto de desarrollo en la planeación de las actividades incluidos los requerimientos, comúnmente experimentaban sobre costos del 40%

al 70% en sus proyectos, en cambio cuando invirtieron de 10% a 20 % del presupuesto destinado al desarrollo en la planeación actividades incluyendo los requerimientos, experimentaron sobrecostos debajo del 30% (Davis, 2005).

El término elicitación se utiliza para resaltar el hecho de que los buenos requisitos no sólo se obtienen desde los clientes, debido a que el cliente no siempre es quien utilizará el sistema. A todas aquellas personas que nos pueden proporcionar requerimientos de un sistema de software se les denomina *stakeholders*. También existe el término *key stakeholder* que son los *stakeholders* que tienen el poder de tomar decisiones y son quienes revisarán los requerimientos, ellos tienen la mayor influencia sobre lo que se hará y lo que no dentro de un proyecto de desarrollo (Phillips, 2009).

Existen tres problemas principales identificados que dificultan la elicitación de requerimientos:

- ⇒ Problemas de alcance: los límites del sistema están mal definidos o los *stakeholders* definen detalles técnicos que confunden más los objetivos del sistema.
- ⇒ Problemas de comprensión: los *stakeholders* no tienen completamente claro lo que se necesita del sistema de software por lo que se les dificulta compartir la información o la consideran obvia.
- ⇒ Problemas de volatilidad: Los requerimientos cambian con el tiempo, puede ser por el crecimiento del sistema o cambio de políticas internas de la empresa que utiliza el sistema.

## **Técnicas de elicitación de requerimientos**

Aunque no existe un método único ni perfecto para la elicitación de requerimientos, las técnicas son distintas en función del proyecto y es función del ingeniero de software encontrar la que mejor se adapte a las necesidades del sistema. Las más utilizadas son las siguientes:

- ⇒ Lluvia de ideas: Realizar sesiones de *brainstorming* para confirmar suposiciones en alto nivel sobre las especificaciones de las necesidades del cliente.

- ⇒ Entrevistas y cuestionarios: Se utilizan para recolectar información, sin embargo, es necesario tener también en cuenta la predisposición del entrevistado, su grado de experiencia o de habilidad, ya que estos elementos tienden a perjudicar la información obtenida durante el proceso a entrevistar.
- ⇒ Historias de usuario: consisten en una aproximación simple a la elicitación de requerimientos que desplaza la importancia desde la formalización de los requerimientos escritos hacia la conversación. Las historias de usuario deberán ser escritas por el cliente, enfatizando aquellas funcionalidades que el sistema deberá realizar.
- ⇒ Talleres: Proporcionan una oportunidad para compartir, refinar y combinar las perspectivas personales de los requerimientos del proyecto.

## **Características óptimas de un requerimiento**

Un requerimiento debe cumplir con atributos tanto grupales como individuales. A continuación se presentan las más importantes:

- ⇒ Necesario: es necesario si su omisión provoca una deficiencia en el sistema y además su capacidad, características físicas o factor de calidad no pueden ser reemplazados por otras capacidades del producto o del proceso.
- ⇒ Conciso: si es fácil de leer y entender. Su redacción debe ser simple y clara para aquellos que vayan a consultarlo en un futuro.
- ⇒ Completo: está completo si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.
- ⇒ Consistente: si no es contradictorio con otro requerimiento.
- ⇒ No ambiguo: cuando tiene una sola interpretación. El lenguaje usado en su definición no debe causar confusiones al lector.
- ⇒ Alcanzable: debe ser un objetivo realista, posible de ser alcanzado con el dinero, el tiempo y los recursos disponibles.
- ⇒ Trazable: se puede determinar el origen de cada requerimiento.

- ⇒ Verificable: cuando puede ser cuantificado de manera que permita hacer uso de los métodos de verificación: interpretación, análisis, demostración o pruebas.
- ⇒ Medible: debe poder medirse objetivamente mediante el conjunto de métricas o indicadores definidos. Estas formas de medición permitirán conocer y validar de forma general y automática el estado del progreso del desarrollo del proyecto.

## **Retos que se presentan en la elicitación de requerimientos**

Los más grandes desafíos a los que se enfrenta una empresa de desarrollo de software, en plena era de la comunicación, según Jaya Choudhury (2013), son los siguientes:

*Comunicación con el equipo a través de las diferentes ubicaciones.* Hoy en día se puede trabajar con personas que se encuentran en otro lugar geográfico y la comunicación para discutir asuntos del proyecto debe planearse con mucho cuidado.

*Revisión de la documentación por parte de los stakeholders antes de la liberación.* Se debe prestar atención en la planificación de una revisión para asegurar la terminación oportuna de los documentos.

*Comprensión de acentos extranjeros,* En resumen, el mercado demanda que el producto este a tiempo, este factor es de suma importancia.

*Creación y mantenimiento de los archivos fuente y de salida.* Es responsabilidad del encargado del levantamiento de requerimientos proveer el formato que el cliente desea y necesita para la documentación de los requerimientos por ejemplo PDF, Word, HTML, etcétera, para visualizarlos en múltiples dispositivos como iPhone, iPad, móviles, etcétera.

*Control de versiones.* Durante el desarrollo del proyecto puede haber más de una persona trabajando en el mismo documento, por lo que es importante definir claramente la línea del proceso que se llevará en la creación y el mantenimiento de los archivos para facilitar la reutilización y el mantenimiento en los diferentes lugares de trabajo.



*Colaboración del equipo de desarrollo y pruebas.* Juega un papel importante durante la fase de entendimiento de los requerimientos, autorización y revisión de procesos. Las líneas divisorias del ciclo de vida del software deben estar bien definidas.

*Traducción de documentos.* Hay aproximadamente 6,500 lenguajes hablados diferentes en el mundo. Cerca del 27 por ciento de la población global habla inglés. Por lo tanto, la necesidad de documentar los requerimientos en distintos lenguajes se incrementa.

*Migración de documentos.* Cuando llega la hora de migrar los documentos se tiene la necesidad de ordenar los estilos de los formatos y plantillas, además de los diferentes estilos de escritura. Cualquier documentación de una organización debe ser consistente y adherirse a los conceptos de paralelismo.

*Multitarea.* En esta era es obligatorio ser parte de múltiples proyectos al mismo tiempo. Se debe estar listo para ponerse en los zapatos de otro, crisis de recursos, actividades no planeadas, etc.

## **Requerimientos de calidad**

Según Chemuturi (2010) en la industria del software las especificaciones se refieren a los requerimientos del usuario final y los posibles escenarios para obtener requerimientos son:

1. Un analista realiza un estudio de viabilidad, redacta un informe y formaliza las necesidades del usuario, a través delo siguiente:
  - ⇒ Se reúne con los usuarios finales, toma nota de sus necesidades y preocupaciones.
  - ⇒ Se reúne con los jefes de departamento y toma nota de sus necesidades y preocupaciones.
  - ⇒ Se reúne con el personal administrativo y toma nota de sus requerimientos y preocupaciones.
  - ⇒ Concreta los requerimientos, los presenta a una selección de usuarios finales, jefes de departamento y administrativos y recibe retroalimentación.
  - ⇒ Implementa la retroalimentación y finaliza las especificaciones.

2. Se prepara una lista de requerimientos y se presenta como parte de una solicitud de propuesta.
3. La solicitud de propuesta apunta a un producto similar y se solicita la replicación con la personalización de las especificaciones del cliente.

Cualquiera que sea el escenario que se presente el objetivo de la calidad en los requerimientos es asegurar que las especificaciones son exhaustivas y que cubren todas las áreas.

Las herramientas para requerimientos de calidad son:

4. Proceso de documentación: Detalla la metodología para la obtención, desarrollo, análisis y finalización de las especificaciones.
5. Estándares y directrices, formatos y plantillas: Especifican el mínimo de especificaciones necesarias para su construcción.
6. Checklist: Ayuda al análisis para asegurar la integridad de las especificaciones.

Al utilizar estas herramientas se pueden desarrollar especificaciones exhaustivas y claras con la calidad necesaria para pasar a la siguiente fase del desarrollo de software (Chemuturi, 2010).

## **Software de calidad**

Chemuturi (2010) pone en relieve la definición de requerimientos de la organización internacional para la estandarización ISO 9000 en su segunda edición, la cual dice que la calidad es un grado en el cual un conjunto de características propias cumplen los requerimientos. En esta definición se pueden apreciar tres palabras clave, los requerimientos que hacen referencia a las propias especificaciones de un producto, las características que hacen al producto apto para su uso y el grado que implica que la calidad continuamente se incrementa de cero al infinito.

En un producto o servicio existen las especificaciones explícitas seleccionadas por el proveedor para ponerlas a disposición del cliente, las implícitas que no se definen pero que se sobreentiende que deberían existir, además existen las especificaciones definidas por un tercero,

que puede ser una oficina gubernamental, una asociación de la industria o un estándar. Las especificaciones formales se convierten en estándares de la industria y son reguladas por asociaciones para el público en general por un precio nominal que cubre el costo de producción y distribución de esos estándares. Ejemplos de asociaciones que regulan estándares son: *American National Standard Institute*, *British Standards Institute*, *Joint Services Specifications*, *ISO*, *Institute of Electrical and Electronics Engineers*, entre otros. Las especificaciones formales o estándares realizan básicamente lo siguiente:

- ⇒ Características de los componentes del producto, como dimensiones, material y métodos de control.
- ⇒ El uso previsto del producto o servicio.
- ⇒ Los lineamientos para la transportación del producto.
- ⇒ Proceso por el que los componentes se fabrican.
- ⇒ Las normas de seguridad y protección que se necesitan en la construcción del producto.

Basándose en lo anterior se puede concluir que la calidad debe ser asegurada por parte del proveedor, por lo cual la calidad debe ser un atributo del producto o servicio ofertado (Chemuturi, 2010).

## **Satisfacción del cliente**

Hablando de calidad el concepto más popular en la industria de la manufactura es el *Total Quality Management* (TQM), ISO define a TQM como un enfoque de gestión para una organización centrada en la calidad, basada en la participación de todos sus miembros con el objetivo de éxito a largo plazo, logrando de la satisfacción del cliente y beneficios para todos los miembros de la organización y la sociedad (Chemuturi, 2010).

En términos generales el cliente quedará satisfecho con un producto de software cuando este cumpla con los requerimientos acordados, es decir el cliente notará la calidad si el software hace lo que el cliente se solicitó.

## Tiempo de entrega

No existe una fórmula exacta para calcular el tiempo que se debe dedicar a la elicitación de requerimientos dentro de un proyecto de desarrollo, debido a que existen muchas razones para la variación (Davis, 2005) por ejemplo:

- ⇒ Problemas no comprendidos requieren más tiempo para entenderlos bien.
- ⇒ Se necesita más tiempo cuando existen varios *stakeholders* con diversas necesidades que cuando existe un solo *stakeholder*.
- ⇒ Los problemas en constante cambio requieren más tiempo que los problemas estáticos.
- ⇒ Problemas complejos requieren más tiempo que los problemas simples.

El tiempo que se asigne a la elicitación del requerimiento va a afectar directamente el tiempo de liberación del requerimiento, es decir a la entrega del producto.

Con un ciclo más corto de entrega de versiones es más probable lograr una entrega a tiempo. Por ejemplo en una compañía de desarrollo en la que el proceso es de un año, se planea satisfacer los requerimientos establecidos en el tiempo de un año, al haber un nuevo requerimiento a mitad del proceso, se tienen dos opciones, una es agregarlos al proceso actual lo que provocara que el tiempo de entrega forzosamente se aplase, la segunda opción es establecer la entrega de ese requerimiento para la siguiente versión, la cual debido a que el ciclo es de un año, tardará un año y medio más el cliente en ver su requerimiento satisfecho lo cual es difícil de aceptar. En cambio, que en un ciclo corto supóngase que es de 3 meses, cuando aparece un nuevo requerimiento a mitad del proceso se evalúa para decidir si se agrega a la versión actual aunque el tiempo de entrega del producto de incremento o si lo planeamos para la siguiente versión que estará lista en 4 meses y medio, esta opción al cliente le parecerá más razonable que la del caso del ciclo largo de un año (Davis, 2005).

## Errores en los requerimientos

Al finalizar un proyecto es difícil determinar cuántos errores se debieron a los requerimientos, no pocos estudios han arrojado que cuando los errores se encuentran ya avanzado el ciclo de vida del proyecto, los costos de detectarlos y repararlos son más significativos que cuando se encuentran los errores en las etapas tempranas del ciclo de vida (Davis, 2005).

A continuación se plantean tres clases de errores que se pueden cometer en los requerimientos según Davis (2005):

1. **Requerimientos desconocidos:** Ocurre cuando el cliente o el equipo de desarrollo desconoce el requerimiento, o puede ser que ambas partes lo desconozcan. Ya sea porque el equipo de desarrollo falló al indagar sobre la información y provoca una pobre elicitación de requerimientos o porque el cliente desconoce la necesidad del negocio. Algunos errores de este tipo se pueden evitar utilizando prototipos al obtener los requerimientos o bien realizando liberaciones en lapsos de tiempo cortos en los cuales los *stakeholders* pueden hacer notar las funcionalidades faltantes o no contempladas. Muchas veces el entregar liberaciones del proyecto en lapsos de tiempo cortos provoca la percepción por parte del equipo de desarrollo de que el cliente jamás estará satisfecho, pero se debe tomar positivamente esta actitud ya que hará que el producto se vaya completando satisfactoriamente.
2. **Errores de planeación:** Ocurren cuando se planea entregar en una versión un requerimiento o un conjunto de requerimientos, pero no se tienen los recursos disponibles suficientes para cubrir el desarrollo. Después de analizar los requerimientos se debe ser realista para visualizar el tiempo, dinero y otros recursos que se necesitaran para satisfacerlos. Repetidas entregas fuera de tiempo son síntomas de una mala planeación.
3. **Errores de especificación:** Ocurren cuando un requerimiento es conocido y correctamente seleccionado e incluido en un entregable, pero es documentado de manera que no está asegurada la comprensión del mismo por todos los involucrados. Esto provoca que el usuario espere una funcionalidad de una manera distinta a como fue implementada por el desarrollador, también ocurre cuando existe ambigüedad en los requerimientos, es decir

que dos desarrolladores entienden cosas distintas de un mismo requerimiento, todo esto provoca la percepción de mala calidad en el producto.

## **Software comercial para la gestión de proyectos**

En el mercado existen varios productos de software para la gestión de proyectos, que pueden ser utilizados por FreeLancer y empresas que manejen una gran cantidad de proyectos, el software que se necesita es un software en línea que se vaya adaptando al posible aumento de proyectos y crecimiento de la empresa, además de que permita la administración de versiones de documentos, existe Microsoft Project uno de los más populares con 22 millones de usuarios y 880 mil clientes, Atlassian JIRA con 30 millones de usuarios y 23 mil clientes, Smartsheet con 1 millón 500 mil usuarios, Basecamp con 3 millones de usuarios, TeamworkPM con 667 mil usuarios (Barrish, 2014).

Un software popular en línea para la gestión de proyectos para empresas pequeñas y medianas es el Teamwork PM, este maneja costos adaptados a la cantidad de proyectos que se manejan, otra opción es el Basecamp pero Teamwork PM tiene varias ventajas que lo hacen más adecuado, entre ellas la navegación intuitiva entre la información contenida, los temporizadores en las tareas, asignación de varias personas en una misma tarea, tareas recurrentes, un panel que muestra los movimientos más recientes, entre otras cosas que podrían ser útiles al administrar los proyectos en línea (Barrish, 2014).

## **Las Pyme en México**

De acuerdo con la Secretaría de Economía (2015) existen dos formas de surgimiento en que surgen las Pequeñas y Medianas Empresas (PYMES), una de ellas son las que se originan como empresas, en las que se puede distinguir claramente una adecuada organización y una estructura, existe gestión empresarial y el trabajo remunerado, éstas en su mayoría se desarrollaron

dentro del sector formal de la economía, por otra parte están aquellas que tuvieron un origen familiar caracterizadas por una gestión que le preocupa su supervivencia sin prestar mucha atención en temas como el costo de oportunidad del capital o la inversión que permite crecimiento.

En México las micro, pequeñas y medianas empresas constituyen un elemento indispensable de la economía nacional por su alto impacto en la generación de empleos y en la producción nacional, según la secretaría de economía las pequeñas y medianas empresas representan en 52% del Producto Interno Bruto (PIB), lo que destaca su importancia en la economía mexicana (Lozano, 2013).

En el estado de Querétaro existe un apoyo para Pymes del sector de la Tecnología de la Información que les permite incorporarse al Programa para el desarrollo de la Industria del Software y las Tecnologías de Información (PROSOFT), este programa ofrece instalación y equipamiento de *Call Centers*, centros de desarrollo, entre otros, así como capacitación técnica y certificaciones para el fortalecimiento de la cadena de valor de la industria del software en Querétaro (Secretaría de Economía , 2015).

Una de las certificaciones que ofrece PROSOFT es en el Modelo de Procesos para la Industria del Software (MOPROSOFT) que en el 2005 se declaró como norma y contiene un modelo creado por la asociación mexicana para a calidad en la ingeniería del software por medio de la Facultad de Ciencias de la Universidad Nacional Autónoma de México (UNAM) y que por solicitud de la Secretaria de Economía provee un modelo adecuado a la mayoría de las empresas mexicanas (IIE, 2003; Promexico, 2013).

La empresa TImobile sobre la cual se realiza esta investigación, obtuvo la certificación MOPROSOFT que, aunque es un modelo adaptado a las empresas mexicanas de desarrollo de software, el modelo en la práctica cotidiana de una Pyme como TImobile no es práctico, es más bien engorroso y difícil de seguir, lo que explica en gran medida, que no se siga en los proyectos de la empresa, pues seguirlo implica una inversión de tiempo importante. Esto hace pensar que fue diseñado para empresas más grandes que TImobile.

Una empresa pequeña a diferencia de una grande, requiere maximizar el tiempo de sus recursos humanos y materiales, para salir adelante con los compromisos, la mayoría de las veces

los recursos humanos son multitarea, es decir, realizan diferentes roles dentro de la empresa, por lo que la documentación de los proyectos se dificulta.



### **III. METODOLOGÍA**

Esta es una investigación cuantitativa y cualitativa de tipo transversal, en la que se compararon dos procesos distintos de elicitación de requerimientos, para realizar esta comparación se estudiaron los módulos del sistema informático Sistema de Control Escolar (SCE) desarrollado por la empresa TImobile.

Uno de los objetivos de esta investigación es el diseño de un proceso simple de elicitación de requerimientos para una Pyme, hecho a la medida de una pequeña empresa de desarrollo de software ubicada en la ciudad de Querétaro.

Una vez diseñado el proceso se probó y se realizaron comparaciones entre ambos procesos de elicitación.

Los dos procesos de elicitación que se compararon fueron el proceso tradicional contra el Proceso Simple de Elicitación de Requerimientos (PROSER) desarrollado durante esta investigación.

Para llevar a cabo esta comparación se dividió la planeación del sistema SCE en dos fases, la primera utilizó el proceso tradicional y la segunda el PROSER. Las fases que utilizaron el proceso tradicional y PROSER contuvieron 19 módulos cada uno.

La población fueron los desarrolladores de software y los usuarios del sistema. El tipo de muestreo para la recolección de los datos fue no probabilístico.

#### **Objeto de estudio**

El objeto de estudio de esta investigación es la comparación entre los dos procesos de desarrollo de software: el proceso tradicional de elicitación de requerimientos en TImobile y el PROSER.

Esta comparación se llevó a cabo dentro del sistema SCE desarrollado para una institución educativa de gran prestigio en la ciudad de Querétaro, la cual tiene una antigüedad en el estado de aproximadamente 25 años, con los niveles de preparatoria y universidad.

El desarrollo del sistema sobre el que se realizarán las comparaciones estuvo planeado a un año y medio, durante el desarrollo del proyecto surgieron varios retos, entre los que destacó la separación del nivel universidad, que la adquirió una empresa educativa española. Al separarse los niveles educativos la universidad sufrió cambios en sus procesos, con ellos cambios de personal con sus propios paradigmas acerca de lo que debería contener cada módulo, lo que implicó más capacitación y nuevas necesidades de información, modificaciones a los requerimientos, nuevas políticas de evaluación, en resumen un sistema muy distinto al planeado en un principio. Otro reto fue la separación física de los dos niveles educativos, es decir, en un principio se tenía toda la información en una misma base de datos con la opción de que el usuario pudiera trabajar con datos de universidad y de preparatoria, al separarse físicamente se definieron usuarios dedicados exclusivamente a cada nivel de la institución.

Al mismo tiempo en que ocurría la separación se desarrollaba el sistema SCE, que era el nuevo sistema implementado para esta institución educativa con la finalidad de renovar, mejorar y hacer uso de las nuevas tecnologías.

Para los fines de esta investigación se dividió en dos fases la evaluación del SCE, la primera con una duración de 8 meses en la que se trabajó con la institución educativa antes de la separación de niveles, en esta fase se trabajó utilizando el proceso tradicional de elicitación de requerimientos utilizado por varios años en TImobile, en la segunda fase se implementó el PROSER (Proceso Simple de Elicitación de Requerimientos) planteado en esta investigación.

Para su estudio y comparación se tomaron 19 de los primeros módulos que se desarrollaron en el proceso tradicional y 19 módulos en el PROSER, fueron seleccionados procurando similitud en tamaño y complejidad para que las comparaciones entre ellos fueran válidas.

El SCE se conforma de tres partes que son determinadas por los usuarios, para lo que es importante definir con claridad cómo se relaciona el sistema con los usuarios dentro del sistema SCE:

- ⇒ Los usuarios Administrativos son aquellos cuyas actividades se relacionan con la administración de la institución como: control escolar, caja, finanzas, registro de cortes bancarios, becas, prórrogas, admisiones y evaluaciones.
- ⇒ Los Usuarios Registrados son los alumnos, padres de familia y profesores.
- ⇒ Los Usuarios Altos Mandos, quienes toman las decisiones económicas y académicas de la institución.

Bajo esta lógica de los usuarios, el SCE se conforma de tres secciones, una sección Windows enfocada a satisfacer las necesidades de los Usuarios Administrativos, otra parte Web que tiene la función de mostrar a los Usuarios Registrados su información de una manera amigable y permitir a los profesores la captura de sus evaluaciones de manera remota, por último una parte móvil enfocada a los Usuarios Altos Mandos dedicada a mostrar indicadores para la toma de decisiones.

Los módulos que se seleccionaron pertenecen en su mayoría a la parte Windows y en menor cantidad de la parte Web por las características que se necesitan para esta investigación.

## **Variables**

Para llegar a cumplir los objetivos de esta tesis y verificar la hipótesis se realizó una simplificación de las variables para facilitar su estudio, así, esta investigación se compuso de tres variables que son: el tiempo de retraso, la cantidad de modificaciones y la satisfacción del usuario.

De la variable *tiempo de retraso* se obtuvo mediante la comparación de la fecha de entrega estimada y la fecha de entrega real, la *cantidad de modificaciones* se obtuvo durante la revisión de

los avances de los módulos con el cliente y finalmente, la variable *satisfacción del usuario* fue obtenida a través de conocer la aceptación del módulo, de esta manera se estima si existió mayor claridad en la comunicación entre el cliente y el desarrollador.

A mayor profundidad, la variable tiempo de retraso se obtuvo al conocer la diferencia entre el tiempo planeado y el tiempo real de entrega, su unidad de medida fueron *días de retraso*. No se contemplaron como días de retraso cuando estos sucedieron por cambios debidos a nuevos requerimientos hechos por el cliente al ver nuevas posibilidades en el sistema, sólo se contabilizaron cuando fueron hechas por malos entendidos en los requerimientos o errores.

Todos los proyectos de desarrollo de software y cada módulo, tienen diferentes características; debido a esto el número de modificaciones aceptables varía, por ello para medir la variable *cantidad de modificaciones* se contabilizó el número de modificaciones solicitadas por el cliente al momento de la primera revisión la entrega final.

La variable *satisfacción del cliente* se obtuvo al contabilizar el número de modificaciones solicitadas por el cliente en la última, se registró un porcentaje de aceptación del cliente en la entrega final, para conocer el grado de satisfacción que percibía en una escala de 0 a 100, en la que se le solicita al desarrollador dar una calificación al módulo entregado, donde 100 significa que el usuario está totalmente satisfecho con la entrega y 0 es la ausencia total de satisfacción.

Los resultados en la satisfacción cualitativa pudieron relacionarse con la personalidad de las personas involucradas, quienes como es natural tenían diferentes puntos de vista y opiniones; entre ellos se encontraban los que aceptaban con entusiasmo al nuevo sistema y los que sólo trataban de encontrar características negativas del sistema, sin embargo centrarnos en este factor impediría ver las sutilezas de las opiniones distintas, por ello sólo se hace mención de este factor para no menospreciarlo en los análisis de los resultados.

## **Proceso de elicitación de requerimientos tradicional en TImobile**

Cómo ya se mencionó el proceso de elicitación de requerimientos *tradicional* fue el que se sometió a comparación con el proceso que se diseñó para esta investigación. A continuación se mencionan las principales características de este sistema tradicional.

El proceso de elicitación de requerimientos que se utilizaba en TImobile consistía en entrevistas con el usuario, se documentaba la entrevista y los requerimientos sin un formato establecido, reunía los documentos proporcionados por el usuario, posteriormente dividía y asignaba las tareas o actividades a los integrantes del equipo de desarrollo a quienes les transmitía verbalmente la información y les entregaba los documentos proporcionados por el usuario.

El equipo de desarrollo trabajaba en el análisis, diseño y desarrollo del proyecto, durante el proceso de desarrollo se disipaban las dudas, en caso de que la duda continuara se llamaba o se visitaba al usuario para que quedara claro el requerimiento.

Una vez terminado un módulo o con un gran avance, se presentaba al usuario el liberable, quien hasta ese momento nos daba la retroalimentación.

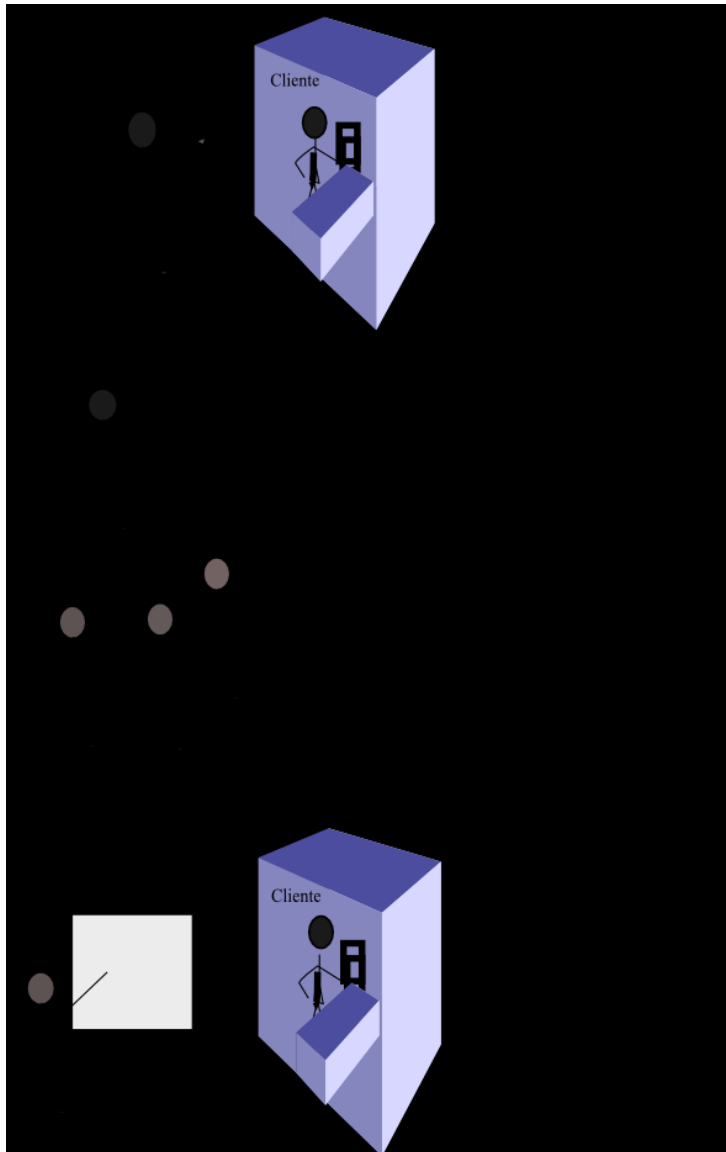
Un entregable puede ser un documento, un avance, un módulo o una sección del proyecto que se acuerda con el cliente para se considere como una entrega, el entregable tiene un tiempo definido para su terminación, cuando se trata de un entregable de desarrollo este debe ser funcional.

Durante la retroalimentación del usuario, al presentarle un avance o al liberar el proyecto, es posible para el equipo saber si se ha estado trabajando bajo los mismos entendidos o de lo contrario se puede enderezar el rumbo para que el proyecto llegue a buen término.

Con el proceso tradicional de elicitación de requerimientos en TImobile, se realizaba la presentación de avances cuando el módulo o proyecto estaba ya casi terminado, es decir que se trabajaba en un proyecto sin corroborar los requerimientos con el usuario y sin asegurarse de que tanto el desarrollador como el usuario hayan entendido lo mismo, esta forma de trabajar la mayoría de las veces arrojaba resultados desfavorables ya que daba la impresión de que no se trabajaba en lo que realmente necesitaba el usuario.

La mayoría de las veces el usuario manifestaba que lo que se le estaba presentando no era lo que había pedido en la entrevista o en el mejor de los casos, manifestaba que faltaban funcionalidades, esto ocurría principalmente por los errores que se cometían al documentar y transmitir los requerimientos a los integrantes del equipo. En TImobile uno de los principales errores era la suposición de requerimientos, es decir, que se creía saber lo que necesitaba el usuario, sin documentarlo y revisarlo junto con el usuario.

El proceso tradicional de elicitación de requerimientos en TImobile se ilustra en la figura 8 donde se puede observar que la retroalimentación sobre el módulo se recibe hasta la presentación del mismo ante el usuario final.



**Figura 6. Proceso de elicitación de requerimientos Tradicional en TMobile.**  
Fuente: Elaboración propia.

Durante la presentación del avance de un módulo o proyecto, el usuario puede ver con claridad lo que necesita realmente, ya que muchas veces el sólo imaginarlo no permite visualizar todas las posibilidades de un sistema informático.

En el proceso tradicional de elicitación de requerimientos de TMobile no se tenía establecido como regla general el acordar revisiones con el cliente, muchas veces debido a la necesidad de entregar el proyecto más rápido y obtener la retribución por el mismo, otras veces porque el cliente está muy ocupado para atender y resolver las dudas del proveedor del sistema.

Este proceso tradicional es atractivo para los clientes porque va implícito un costo menor ya que se evita el análisis y se va directamente al desarrollo y tiempo de re programación lo asume la empresa TImobile.

A pesar de las numerosas modificaciones sobre un entregable casi terminado, generalmente se llegaba a un buen término del proyecto.

## **Diseño y comparación con el proceso PROSER**

Dentro de la empresa de TImobile, la autora de esta investigación lleva laborando alrededor de 10 años lo que la ha brindado la habilidad de detectar y conocer algunas áreas de oportunidad de la empresa, en este sentido el diseño de la plantilla SRS de PROSER fue diseñada para responder a la necesidad de simplificar el proceso de levantamiento de requerimientos en la empresa.

Así, gracias al tiempo laborando en la empresa y mientras estudiaba la Maestría en Tecnologías de la Información fue claro que en la empresa TImobile existían una cantidad considerable de retrasos en la entrega de proyectos y liberables, que gastaban recursos de la empresa, situación que se tornaba recurrente y que podía mejorar con la intervención adecuada.

Se encontró también, cierta inconformidad generalizada en el área de desarrollo ya que los integrantes de los equipos dijeron no tener certeza, cuando se desarrolla un proyecto, de estar haciendo lo pidió el cliente al comienzo, esto debido que no se tiene documentación suficiente de los requerimientos, la mayoría de las veces se dan cuenta de que se trabaja en lo correcto cuando se le muestra un avance o el prototipo al cliente, si no es lo que requería el cliente hay que trabajar en las modificaciones necesarias para encausar el proyecto adecuadamente lo que implica tiempo y retrasos en la entrega, esto genera insatisfacción en el cliente, quien percibe que el proyecto no se desarrolla con calidad.

Al analizar el proceso actual de levantamiento o elicitación de requerimientos en el proceso de desarrollo de TImobile se consideró la posibilidad de que si se comenzaban a estructurar los cimientos del proceso de desarrollo se conseguirían mejores resultados.

Así se realizó el diseño del PROSER.



Y la segunda fase del proyecto del SCE, se desarrolló basado en el proceso PROSER. Esta segunda fase tuvo una duración de 7 meses y se trabajó utilizando el proceso diseñado en esta investigación.

Se compararon los elementos descritos en las variables.

## IV. RESULTADOS

El diseño del proceso PROSER se llevó a cabo por medio del análisis de la literatura disponible, misma que ha sido mencionada en esta tesis, además, se tomó en cuenta la experiencia de 10 años de la autora en el desarrollo de software para esta empresa y el conocimiento en proyectos pasados, así como de todo tipo de experiencias con los clientes, experiencias vividas y compartidas en el equipo de trabajo.

### **Diseño del proceso simple para la elicitación de requerimientos (PROSER)**

El proceso que se diseñó para esta investigación consta de 4 documentos tipo plantilla:

- ⇒ Documento de visión: Es un documento libre en el que se documenta varios puntos de un proyecto como el alcance, el problema que va a solucionar, a quién va enfocado y varios puntos más, la idea es que se documente en lo posible la mayor parte de este documento al menos una primera vez después según lo permita el proyecto se podrán hacer revisiones a este mismo documento (Anexo B).
- ⇒ Documento de necesidades del cliente: En el que se enlistan las necesidades de los stakeholders y que se espera queden cubiertos con el sistema informático dándole una prioridad numérica a cada una donde 10 es de baja prioridad y 1 es de alta prioridad, este documento se debe llenar junto con el cliente para que sea él quien especifica las prioridades. Además en este documento se pueden registrar las fechas de entrega de avances o liberables (Anexo C).
- ⇒ Plantilla de documentación de requerimientos: Es un documento diseñado por la autora de esta investigación basándose en el estándar IEEE, en donde se documentan los requerimientos del sistema informático. Se deberá tener un documento por cada funcionalidad del proyecto (Anexo D).

⇒ Plantilla para la documentación: Diseñado por la autora de esta investigación basándose en la experiencia, en el que se documentara de manera general o específica el funcionamiento de un módulo o proceso dentro del sistema informático, este documento se llena al tener terminado y entregado un módulo y facilitará la creación del manual de usuario (Anexo E).

La parte más importante de PROSER es la documentación de requerimientos que se refiere al llenado del formato de requerimientos, en el cual se describen las necesidades de los stakeholders en un módulo o proceso, esta descripción no debe contener detalles de diseño simplemente se describirá lo que se espera del sistema, usando el siguiente formato:

<<*objeto o función*>> + deberá + <<*acción a realizar*>> + <<*condiciones*>>

O bien,

Se deberá poder + <<*acción a realizar*>> + <<*objeto o función*>> + <<*condiciones*>>

Por ejemplo:

“Al guardar deberá crear un usuario y una contraseña encriptada y enviarla al correo del destinatario cuando haya sido exitoso el proceso de guardado”.

O bien,

“Se deberá enviar un correo electrónico con los datos de usuario y contraseña al guardar cuando el guardado haya sido exitoso”.

La plantilla para la documentación de los requerimientos, entre otros, está basada en el estándar IEEE, la plantilla debe ser llenada por el levantador de los requerimientos o entrevistador y revisada en conjunto con el cliente para que se realicen las modificaciones necesarias, antes de comenzar con el desarrollo del entregable.

El objetivo del llenado de la plantilla es, que sea un entregable de desarrollo, antes de la entrega final de un requerimiento se deben realizar 3 revisiones con un lapso máximo de 2 semanas.

La primera revisión servirá como indicador de que se entendieron los requerimientos de parte del equipo de trabajo y para clarificar dudas que se tengan acerca del entregable, dependiendo del tamaño y complejidad del requerimiento desde la primera revisión pueden mostrarse pantallas e incluso funcionalidades.

La segunda o tercera revisión es la de la entrega definitiva pero se pueden tener las revisiones necesarias, siempre y cuando se documenten apropiadamente, para que la documentación sirva como justificación si el tiempo de entrega se llega a aplazar, además de que se puede estudiar la maduración de los requerimientos a través del tiempo. La figura 8 muestra el proceso que se realiza con PROSER.

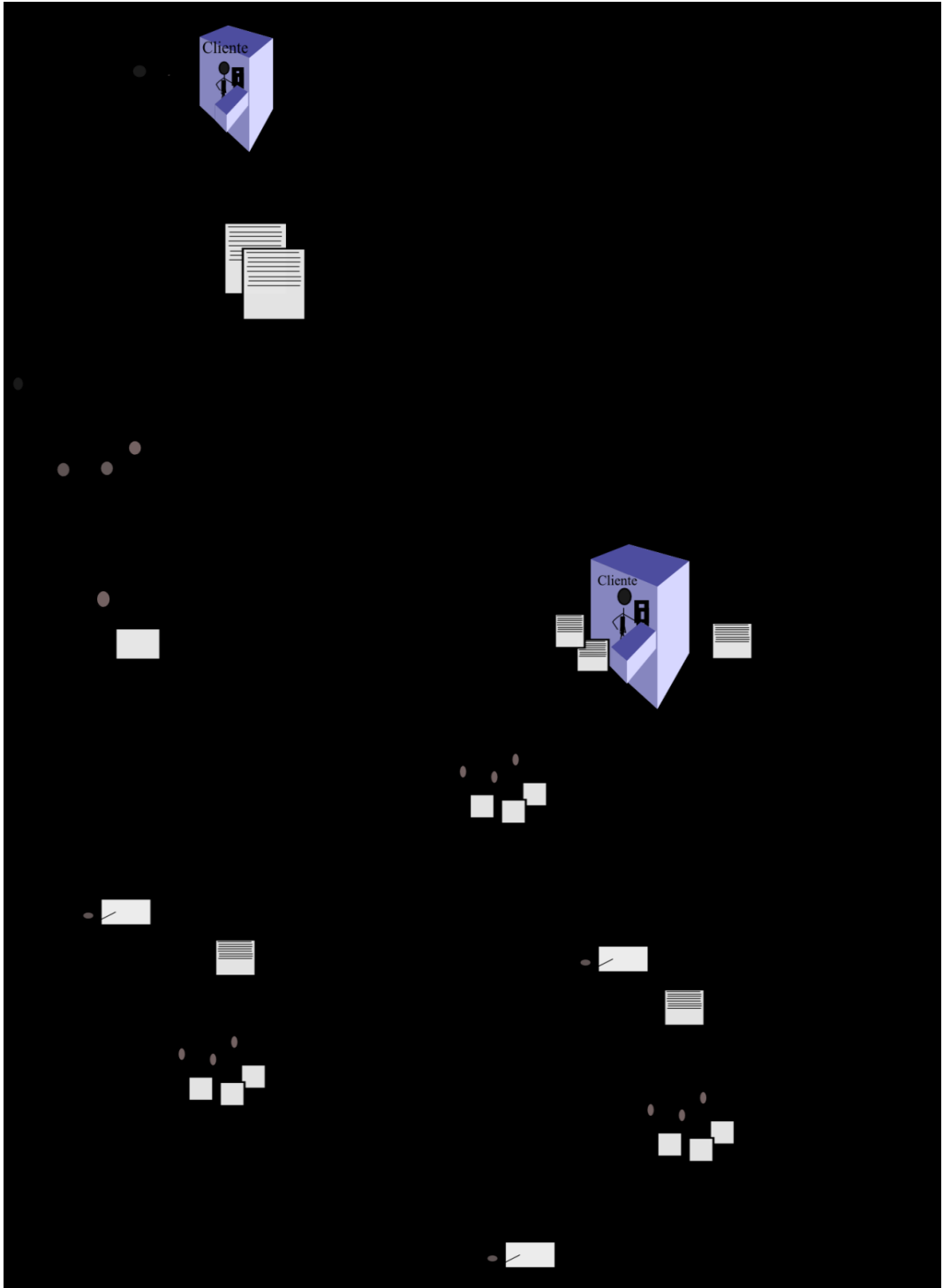


Figura 7. Proceso Simple de Elicitación de Requerimientos PROSER en contacto con el cliente.

Fuente: Elaboración propia.

En PROSER existen 2 etapas de documentación la primera se realiza en contacto con el cliente como lo muestra la figura anterior, el listado de las necesidades del cliente, el llenado del formato de requerimiento, la validación con el cliente y las presentaciones tienen que involucrar sin ninguna duda al cliente.

En la segunda etapa de documentación se planeó llenar una plantilla diseñada para que permitiese tener documentado el funcionamiento del sistema y facilitara la creación del manual del usuario.

Asimismo, la creación del manual de usuario es sencilla pero se tiene que redactar en términos que cualquier persona pueda comprender, no se valen tecnicismos, este manual se introduce en una segunda etapa de documentación porque la plantilla para la documentación (Anexo E), se llena una vez que ya no se tienen modificaciones sobre el requerimiento, el proceso se ilustra en la figura 10.

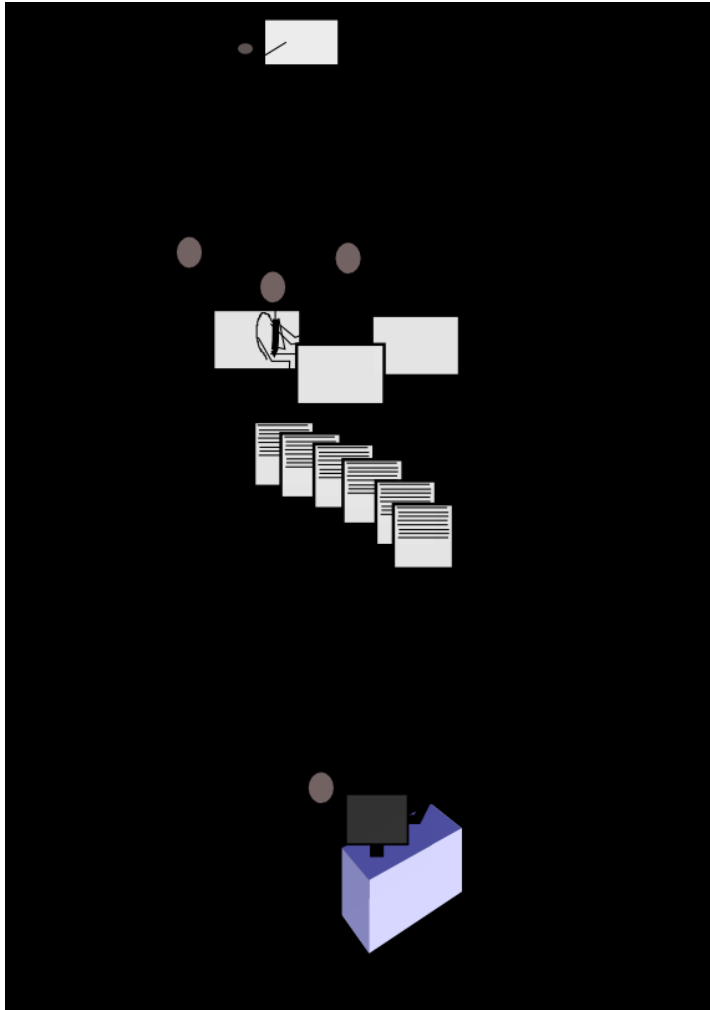


Figura 8. PROSER etapa de documentación para el manual.  
Fuente: Elaboración propia.

## Creación de la plantilla SRS de PROSER

Investigando y estudiando la ingeniería de requerimientos se encontró que existen varios estándares entre ellos el de IEEE Std. 830-1998 (ver en Anexo A) que tiene gran aceptación en el campo del desarrollo de software y que es muy completo y se puede aplicar de manera libre.

La plantilla mostrada en el anexo A es muy completa, pero en el caso de los proyectos de TImobile varios espacios quedan en blanco debido a que se va conociendo la información conforme se avanza en el proceso, así se diseñó una plantilla a la medida, que se adaptase a las necesidades

actuales y reales de la empresa, que contuviese datos básicos que necesita el desarrollador de software para realizar su labor con certidumbre y con un documento claro que respalde su trabajo.

Siempre contemplando dos realidades básicas, las necesidades de la empresa de ahorro en tiempo para seguir en la competencia y la posibilidad de ahorrar tiempo de reprogramación invirtiendo adecuadamente el tiempo en llevar a cabo un proceso adaptado a la medida.

Incluyendo estos elementos se diseñó la plantilla para la documentación de los requerimientos tomando como base la especificación de requerimientos y la asignación de prioridades de la plantilla IEEE std. 830-1998, admitiendo y adaptando los elementos que dan funcionalidad, ahorro de tiempo e inversión apropiada para el tamaño de una Pyme y excluyendo aquellos que requieren un alto gasto de tiempo y energía, diseñados para grandes corporaciones.

Se tomó la decisión de descartar una gran cantidad de apartados, que por el tamaño de la empresa y las necesidades no son viables, rescatando y modificando las que documentan lo esencial para un correcto proceso de desarrollo. La premisa básica para este proceso de exclusión e inclusión se basó en el conocimiento de la empresa TImobile, adquirido en 10 años de experiencia, en los proyectos anteriores, los principales errores del pasado y presente, así como las mayores recurrencias que han supuesto gastos innecesarios para la empresa.

Además, se sustituyeron elementos que representaban gran gasto en tiempo por otros con los que se obtienen los mismos resultados y permiten a una Pyme ahorrar tiempo y esfuerzo a la vez de ayudar a documentar adecuadamente, así, en lugar de funciones de producto se enlistaron una secuencia de pasos que deben seguirse para el adecuado funcionamiento de la especificación en el sistema.

En este mismo sentido se sustituyeron los elementos de requerimientos de performance por una resumida secuencia de excepciones que permite identificar las posibles anomalías o irregularidades en el rendimiento de manera rápida y eficaz.



## Tiempo de retraso: Tradicional Vs PROSER

Recordando que en cada proceso se trabajó con distintos módulos, estos fueron equiparables en tamaño, quedando de la siguiente forma:

	#	Módulos proceso Tradicional
Grande	1	Aspirantes
Grande	2	Consulta de estudiantes
Grande	3	Caja
Grande	4	Planes de pago
Grande	5	Planes de estudio
Grande	6	Control de becas
Grande	7	Control de calificaciones
Mediano	8	Becas de admisiones
Mediano	9	Inscripción y re-inscripción
Mediano	10	Listas y actas
Mediano	11	Corte bancario
Mediano	12	Cobro de extraordinarios
Mediano	13	Facturación Preparatoria
Mediano	14	Facturación Universidad
Chico	15	Reportes de aspirantes
Chico	16	Creación de grupos
Chico	17	Búsqueda de estudiantes
Chico	18	Generación recibos pago
Chico	19	Reportes de evaluaciones

**Tabla 1. Módulos proceso tradicional**

	#	Módulos proceso PROSER
Grande	20	Estado de cuenta
Grande	21	Acuerdos y prorrogas
Grande	22	Examen de admisión
Grande	23	Presentación de examen en Web
Grande	24	Expediente tutorías
Grande	25	Plantilla Dossier (Web)
Grande	26	Expediente profesores
Mediano	27	Bitácora
Mediano	28	Reportes de pagos
Mediano	29	Evaluación a profesores
Mediano	30	Autoevaluación de profesores
Mediano	31	Control de documentos
Mediano	32	Expediente de egresados/titulación
Mediano	33	Expediente de Inglés
Chico	34	Recepción de documentos
Chico	35	Planeación de periodos
Chico	36	Expediente clínico
Chico	37	Expediente Servicio social
Chico	38	Expediente Prácticas profesionales

**Tabla 2. Módulos proceso PROSER**

En las tablas, tabla 1 y tabla 2, se muestra cómo se dividieron para su comparación los módulos que se desarrollaron con el proceso tradicional y con el sistema PROSER. Se dividieron en 7 grandes, 7 medianos y 5 chicos para cada proceso, se encuentran acomodados de grandes a chicos, si bien no son iguales 1 a 1, es posible interpretar de manera equivalente todos los grandes, todos los medianos y todos los chicos de ambos procesos.

El tiempo de retraso se obtuvo a partir de los días extra que se requirieron para completar cada módulo. Ahora bien, la comparación del *tiempo de retraso* se hizo de dos maneras, la primera comparación es:

*Proceso tradicional: Tiempo estimado Vs tiempo de retraso*

Y la segunda comparación fue:

*PROSER: Tiempo estimado Vs tiempo de retraso*

Así, la tabla3 contiene información sobre la primer comparación, en la primera columna se observa la inicial del tamaño del módulo para grande (G), para mediana (M) y para chico (CH). En la gráfica1 las primeras líneas corresponden a los mayores disminuyendo hacia la derecha.

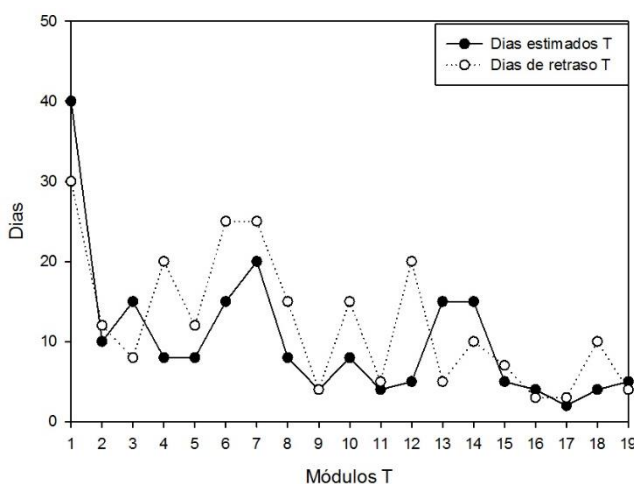
En la segunda columna de la tabla3 se observa el número del módulo, que corresponde en la gráfica.

En la cuarta columna de la tabla3 está el estimado de la fecha en la empresa se comprometió a entregar el módulo al cliente, 40 días, sin embargo se entregó 30 días después.

La gráfica1 muestra los datos de la tabla3, los días estimados se encuentran señalados con una línea negra y los días de retraso están indicados con una línea punteada. Se observa en X los módulos del proceso tradicional y en Y los días.

	#	Módulo Tradicional	Días estimados	Días retraso
G	1	Aspirantes *	40*	30
G	2	Consulta de estudiantes	10	12
G	3	Caja	15	8
G	4	Planes de pago	8	20
G	5	Planes de estudio	8	12
G	6	Control de becas	15	25
G	7	Control de calificaciones	20	25
M	8	Becas de admisiones	8	15
M	9	Inscripción y re-inscripción	4	4
M	10	Listas y actas	8	15
M	11	Corte bancario	4	5
M	12	Cobro de extraordinarios	5	20
M	13	Facturación Preparatoria	15	5
M	14	Facturación Universidad	15	10
CH	15	Reportes de aspirantes	5	7
CH	16	Creación de grupos*	4*	3*
CH	17	Búsqueda de estudiantes*	2*	3*
CH	18	Generación recibos pago	4	10
CH	19	Reportes de evaluaciones	5	4

Tabla 3. Días de retraso proceso tradicional



Gráfica 1. Tradicional, días estimados Vs días de retraso

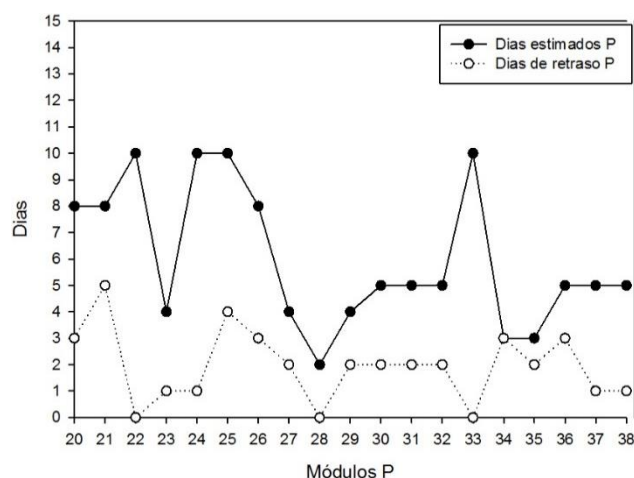
La escala de los días de retraso oscila ente 4 días y 30 días, es visible que los proyectos grandes tomaron mayor cantidad de tiempo. El módulo que más días de retraso tuvo fue el 1 de Aspirantes que se estimó en 40 días y tuvo 30 días de retraso\*. Los que mejor se estimaron fueron el 16 y 17, Creación de grupos y Búsqueda de estudiantes, ambos con sólo 3 días de retraso\*. No hubo ninguno con cero días de retraso.

La tabla 4 contiene la relación entre el número en que es graficado cada módulo y el nombre, los días estimados y los días de retraso, muestra también el tamaño de los módulos.

En la gráfica 2 es posible ver los días estimados y los días de retraso del proceso PROSER, la escala de la gráfica oscila entre 0 y 15 días. Se observa en **X** los módulos de PROSER y en **Y** los días que transcurrieron, en la línea negra los días estimados y en la línea punteada los días de retraso.

	#	Módulo PROSER	Días estimados	Días retraso
G	20	Estado de cuenta	8	3
G	21	Acuerdos y prorrogas	8	5
G	22	Examen de admisión*	10	0
G	23	Presentación de examen en Web	4	1
G	24	Expediente tutorías	10	1
G	25	Plantilla Dossier (Web)*	10	4
G	26	Expediente profesores	8	3
M	27	Bitácora	4	2
M	28	Reportes de pagos*	2	0
M	29	Evaluación a profesores	4	2
M	30	Autoevaluación de profesores	5	2
M	31	Control de documentos	5	2
M	32	Expediente de egresados/titulación	5	2
M	33	Expediente de Inglés*	10	0
CH	34	Recepción de documentos	3	3
CH	35	Planeación de periodos	3	2
CH	36	Expediente clínico	5	3
CH	37	Expediente Servicio social	5	1
CH	38	Expediente Prácticas profesionales	5	1

Tabla 4. PROSER, días estimados Vs días retraso



Gráfica 2. PROSER, días estimados Vs días de retraso

En el PROSER no se observa a simple vista una relación entre el tamaño del módulo con los días de retraso. El módulo que más días de retraso tuvo fue el 25, corresponde a la Plantilla Dossier y fue planeada para 10 días y tuvo apenas 4 días de retraso\*. Los módulos con menos días de retraso fueron: 22 Examen de Admisión, 28 Reportes de pagos y 33 Expediente de inglés, fueron estimados para su desarrollo con 10, 2 y 10 días respectivamente y los tres no tuvieron un solo día de retraso; fueron entregados a tiempo\*.

## Días de retraso Vs días totales

Otra manera de comparar los datos arrojados por los días de retaso es mediante la comparación entre:

*Proceso tradicional: Tiempo estimado Vs días totales y*

*PROSER: Tiempo estimado Vs días totales*

Siendo los días totales la suma entre los días estimados y los días de retraso.

	#	Módulo Tradicional	Días estimados	Días Totales
G	1	Aspirantes	40	70
G	2	Consulta de estudiantes	10	22
G	3	Caja	15	23
G	4	Planes de pago	8	28
G	5	Planes de estudio	8	20
G	6	Control de becas	15	40
G	7	Control de calificaciones	20	45
M	8	Becas de admisiones	8	23
M	9	Inscripción y re-inscripción	4	8
M	10	Listas y actas	8	23
M	11	Corte bancario	4	9
M	12	Cobro de extraordinarios	5	25
M	13	Facturación Preparatoria	15	20
M	14	Facturación Universidad	15	25
CH	15	Reportes de aspirantes	5	12
CH	16	Creación de grupos	4	7
CH	17	Búsqueda de estudiantes	2	5
CH	18	Generación recibos pago	4	14
CH	19	Reportes de evaluaciones	5	9

Tabla 5. Tradicional, días estimados Vs días totales

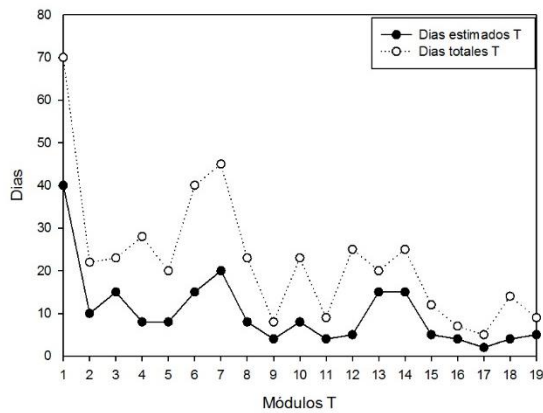
	#	Módulo PROSER	Días estimados	Días Totales
G	20	Estado de cuenta	8	11
G	21	Acuerdos y prorrogas	8	13
G	22	Examen de admisión	10	10
G	23	Presentación de examen en Web	4	5
G	24	Expediente tutorías	10	11
G	25	Plantilla Dossier (Web)	10	14
G	26	Expediente profesores	8	11
M	27	Bitácora	4	6
M	28	Reportes de pagos	2	2
M	29	Evaluación a profesores	4	6
M	30	Autoevaluación de profesores	5	7
M	31	Control de documentos	5	7
M	32	Expediente de egresados/titulación	5	7
M	33	Expediente de Inglés	10	10
CH	34	Recepción de documentos	3	6
CH	35	Planeación de periodos	3	5
CH	36	Expediente clínico	5	8
CH	37	Expediente Servicio social	5	6
CH	38	Expediente Prácticas profesionales	5	6

Tabla 6. PROSER, días estimados Vs días totales

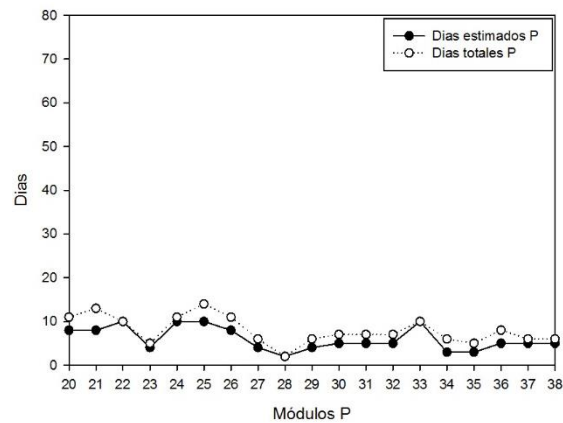
La tabla 5 y tabla 6 contienen los tamaños de los módulos, la numeración con que se grafican, los días estimados y los días totales que se requirieron para terminar el módulo. La tasa de retraso PROSER= 24%, mientras que la tasa de retraso tradicional = 54% .

Entonces la tasa de retraso del proceso tradicional bajó del 54% al 24% con el nuevo proceso, lo que demuestra que sí bajó el tiempo de retraso con el proceso PROSER.

En la gráfica 3 y la gráfica 4 se muestra otra comparación, en **X** los módulos de cada proyecto y en **Y** los días, con línea negra se muestran los días estimados, mientras que con una línea punteada se muestran los días totales en que se trabajó en cada módulo.



Gráfica 4. Tradicional, días estimados Vs días totales



Gráfica 3. PROSER, días estimados Vs días totales

En estas gráficas es evidente el cambio que existió en los tiempos de retraso con cada uno de los procesos, mientras que en el proceso tradicional el tiempo más alto total fue de 70 días, el más alto del PROSER fue de 14 días, esto puede interpretarse de manera inadecuada, pues posiblemente fue un módulo más complicado y con características esencialmente distintas, por lo que se sugiere discreción al momento de sobre interpretar. Todos los módulos tienen diferente dificultad.

Sin embargo en los otros módulos, es evidente que las líneas de la grafica4 de PROSER están en general ambas líneas más cercanas entre sí.

La gráfica 5 muestra la comparación entre los dos procesos, la comparación de las medias de días de retraso, en los dos procesos. Los puntos muestran los días de retraso en cada uno de los módulos, los puntos azules muestran dónde se ubica la media.



	#	Módulo Tradicional	Inicial	Fin
G	1	Aspirantes	10	6
G	2	Consulta de estudiantes	9	8
G	3	Caja**	30*	10*
G	4	Planes de pago	15	7
G	5	Planes de estudio	25	5
G	6	Control de becas**	30*	8
G	7	Control de calificaciones**	30*	10*
M	8	Becas de admisiones	17	10*
M	9	Inscripción y re-inscripción	5	5
M	10	Listas y actas	20	6
M	11	Corte bancario	8	5
M	12	Cobro de extraordinarios	17	6
M	13	Facturación Preparatoria	10	6
M	14	Facturación Universidad	20	4
CH	15	Reportes de aspirantes	4	5
CH	16	Creación de grupos	10	5
CH	17	Búsqueda de estudiantes	3	3
CH	18	Generación recibos pago	20	8
CH	19	Reportes de evaluaciones	5	3

Tabla 7. Modificaciones proceso tradicional

	#	Módulo PROSER	Inicial	Fin
G	20	Estado de cuenta	3	2
G	21	Acuerdos y prorrogas	10	1
G	22	Examen de admisión	5	2
G	23	Presentación de examen en Web	2	1
G	24	Expediente tutorías	6	3
G	25	Plantilla Dossier (Web) *	11*	4
G	26	Expediente profesores	5	3
M	27	Bitácora	3	3
M	28	Reportes de pagos	2	0
M	29	Evaluación a profesores*	3	5*
M	30	Autoevaluación de profesores	6	3
M	31	Control de documentos	5	1
M	32	Expediente de egresados/titulación	1	1
M	33	Expediente de Inglés	0	4
CH	34	Recepción de documentos	5	3
CH	35	Planeación de periodos	5	1
CH	36	Expediente clínico	6	2
CH	37	Expediente Servicio social	10	1
CH	38	Expediente Prácticas profesionales	0	2

Tabla 8. Modificaciones PROSER

Respecto las modificaciones iniciales el número más alto en el tradicional fue de 30, y ocurrieron en los módulos 3 Caja, 6 Control de becas y 7 Control de calificaciones, mientras que el número más alto de modificaciones iniciales en el PROSER fue de 5, en el 25 Plantilla Dossier\*.

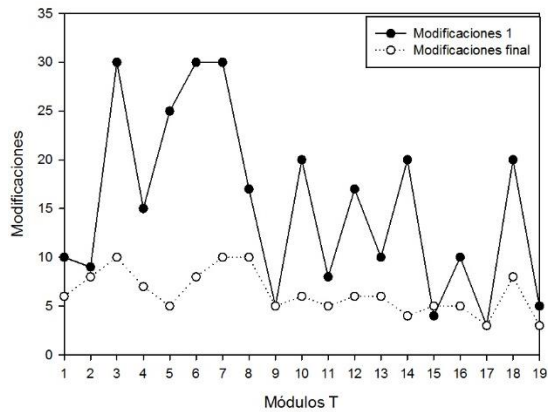
Respecto a las modificaciones finales el número más alto en el tradicional fueron 10 en los módulos 3 Caja, 6 Control de becas y 7 Control de calificaciones, los mismos que en los iniciales, mientras que en el PROSER fueron 5 modificaciones, en la 29 Evaluación a profesores\*.

Existen múltiples formas para graficar estas tablas, se decidió realizar las siguientes comparaciones: del mismo proceso, 1) Modificaciones iniciales Tradicional vs modificaciones finales Tradicional y 2) Modificaciones iniciales PROSER vs modificaciones finales PROSER. Comparar entre procesos: 1) Modificaciones iniciales Tradicional vs modificaciones iniciales PROSER y 2) Modificaciones finales Tradicional vs modificaciones finales PROSER.

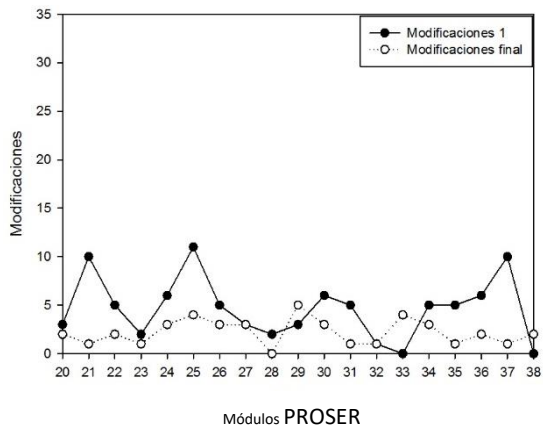
En la grafica 6 se muestra la comparación entre los módulos desarrollados con el proceso Tradicional y en la gráfica 7 se muestran los módulos desarrollados con el PROSER.

En la absisa de la X se encuentran los módulos que se desarrollaron con cada proceso y en la ordenada de la Y se encuentran el numero de modificaciones que se hicieron en cada etapa.

El línea recta se muestran las modificaciones iniciales, mientras que en línea punteda se observan las modificaciones finales.



Gráfica 6. Modificaciones proceso tradicional



Gráfica 7. Modificaciones PROSER

Es posible notar a simple vista que las modificaciones en el proceso tradicional fueron mucho más elevadas en comparación con el PROSER. En las iniciales, el proceso tradicional llegando a 30 mientras que en PROSER llega sólo a 10. Además es visible que en las modificaciones finales las más altas del tradicional llegan a 10 y en PROSER llegan a cero modificaciones. Con el proceso tradicional, en ninguna de las dos entregas hubo cero modificaciones mientras que en PROSER hubo cero modificaciones en un módulo inicial y un módulo final. En los módulos desarrollados con el proceso Tradicional oscila la escala entre 3 y 30 modificaciones en ambas revisiones. En los módulos desarrollados con el PROSER la escala oscila entre 0 y 11 modificaciones en ambas revisiones.



En el proceso tradicional la media de modificaciones iniciales  $\bar{x}=15$ , mientras que la media de las modificaciones finales fue de  $\bar{x}=6$ . La desviación estándar en las modificaciones iniciales es de  $\delta=9$  mientras que en las modificaciones finales es de  $\delta=2$ .

Análogamente en el PROSER la media de las modificaciones iniciales es de  $\bar{x}=4$ , mientras que en las modificaciones finales fue de  $\bar{x}=2$ . La desviación estándar en las modificaciones iniciales es de  $\delta=3$  y en las modificaciones finales  $\delta=1.3$ .

Ahora bien, también es posible comparar entre procesos, con la dificultad de que los módulos son todos distintos y contienen características distintas entre sí, como ya se mencionó se realizó una equiparación en cuestiones de tamaño y dificultad.

La tabla 9 ejemplifica la forma en que se organizaron los datos para hacer esta comparación.

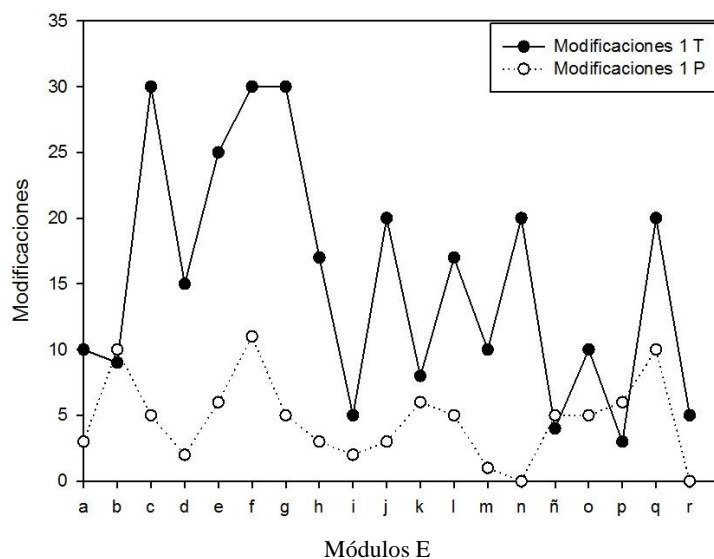
Muestra cómo se nombraron los conjuntos de datos, para organizarlos y poder comparar entre dos distintos procesos, comparar las modificaciones iniciales y las modificaciones finales.

	#	Módulo Tradicional		Módulo PROSER	#	
G	1	Aspirantes	<b>a</b>	Estado de cuenta	20	G
G	2	Consulta de estudiantes	<b>b</b>	Acuerdos y prorrogas	21	G
G	3	Caja	<b>c</b>	Examen de admisión	22	G
G	4	Planes de pago	<b>d</b>	Presentación de examen en Web	23	G
G	5	Planes de estudio	<b>e</b>	Expediente tutorías	24	G
G	6	Control de becas	<b>f</b>	Plantilla Dossier (Web) *	25	G
G	7	Control de calificaciones	<b>g</b>	Expediente profesores	26	G
M	8	Becas de admisiones	<b>h</b>	Bitácora	27	M
M	9	Inscripción y re-inscripción	<b>i</b>	Reportes de pagos	28	M
M	10	Listas y actas	<b>j</b>	Evaluación a profesores*	29	M
M	11	Corte bancario	<b>k</b>	Autoevaluación de profesores	30	M
M	12	Cobro de extraordinarios	<b>l</b>	Control de documentos	31	M
M	13	Facturación Preparatoria	<b>m</b>	Expediente de egresados/titulación	32	M
M	14	Facturación Universidad	<b>n</b>	Expediente de Inglés	33	M
CH	15	Reportes de aspirantes	<b>ñ</b>	Recepción de documentos	34	CH
CH	16	Creación de grupos	<b>o</b>	Planeación de periodos	35	CH
CH	17	Búsqueda de estudiantes	<b>p</b>	Expediente clínico	36	CH
CH	18	Generación recibos pago	<b>q</b>	Expediente Servicio social	37	CH
CH	19	Reportes de evaluaciones	<b>r</b>	Expediente Prácticas profesionales	38	CH

Tabla 9. Organización de módulos para su comparación

Los datos de la tabla 9 son graficados en la gráfica 8, en la cual se comparan las modificaciones iniciales desarrolladas con el proceso Tradicional, en la línea recta y las modificaciones iniciales desarrolladas con PROSER en la línea punteada.

En la **X** se encuentran los módulos desarrollados y ordenados según los criterios de la tabla 9, nombrados *Módulos E*, en la **Y** se encuentran los número de modificaciones que se realizaron.



**Gráfica 8. Modificaciones iniciales tradicional Vs PROSER**

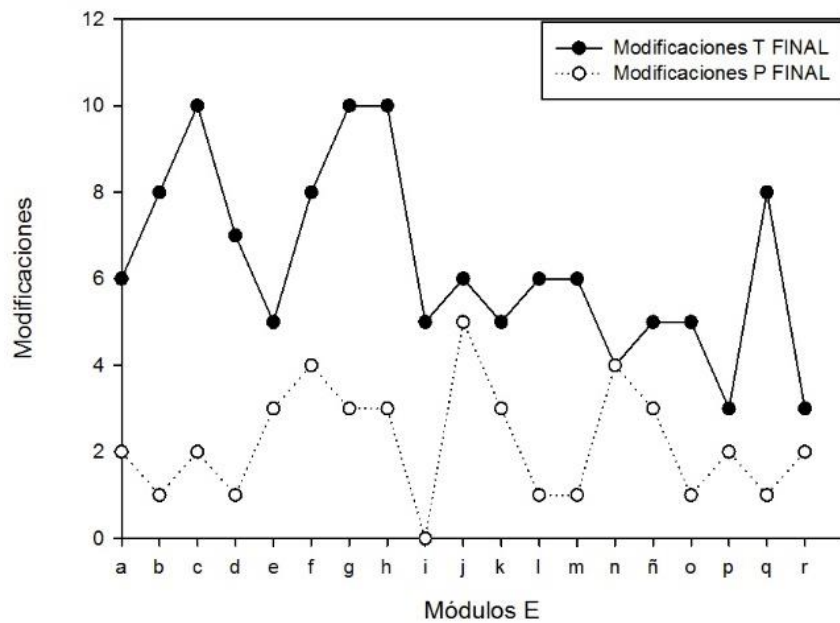
Es observable en la gráfica 8 la diferencia entre el número de modificaciones en cada uno de los dos procesos.

Es claro que las modificaciones iniciales con el proceso tradicional son muchas más que en el PROSER, esto repercute enormemente en el clima que se vive con el cliente, la relación que se establece ante una primera revisión con tantas modificaciones. Con este grado de modificaciones, es posible que el cliente tenga la sensación de que no fueron comprendidos los requerimientos o fueron mal interpretados.

Otra de las repercusiones en de tantas modificaciones en la primera entrega es la frustración de los desarrolladores, imaginarse en su situación de tener tal número de modificaciones, tan sólo al inicio, puede saber que aumentarán los retrasos, aumentarán las complicaciones y la ambigüedad de buscar ahora los cambios necesarios al código, a sabiendas de todo desarrollador que cambios drásticos implican algunas veces que sea más sencillo iniciar de nuevo el código que cambiar la lógica del que ya se tiene.

En ese mismo orden se encuentra la gráfica 9, en esta se muestra la comparación de las modificaciones finales en los distintos procesos. En la línea recta se muestra el proceso tradicional

y en la línea punteada el proceso PROSER, en la X se muestran los módulos denominados E (tabla 9) y en el eje de la Y el número de modificaciones.



Gráfica 9. Modificaciones finales proceso tradicional Vs PROSER

Lo que se puede comprender al observar esta gráfica es que en la recta final del proceso, existieron menos modificaciones al código con PROSER, como hubo un mayor número de revisiones podría ser lógico que el cliente tuvo más oportunidades para verificar si el módulo contenía lo que él había solicitado, así como que desde el inicio tuviese la oportunidad de verificar si el sistema contendría lo que él solicitaba.

### Porcentaje de aceptación Tradicional Vs PROSER

El porcentaje de aceptación es un dato que dio el desarrollador según el percibió la aceptación del cliente en la entrega final.

A continuación se muestra la tabla 10 y la tabla 11 con los porcentajes de aceptación de los distintos módulos tanto en el proceso tradicional como en PROSER.

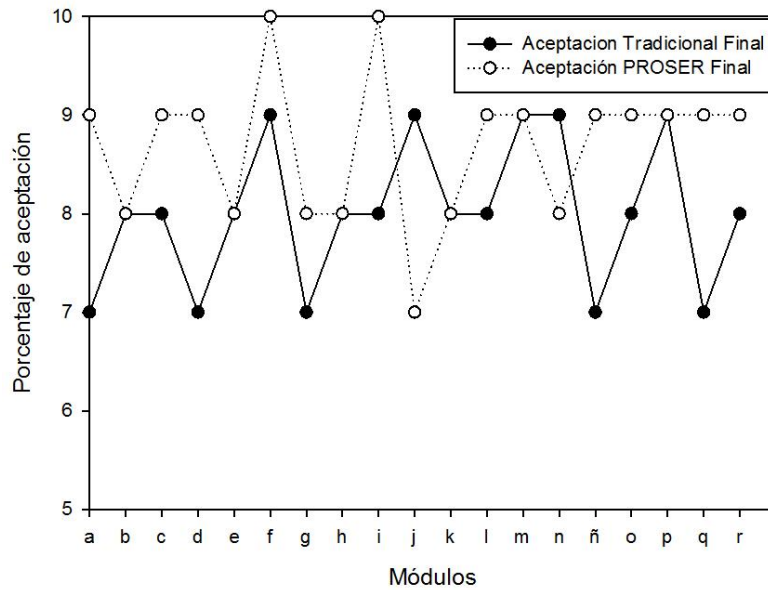
	#	Módulo Tradicional	Porcentaje de aceptación
G	1	Aspirantes	70
G	2	Consulta de estudiantes	80
G	3	Caja	80
G	4	Planes de pago	70
G	5	Planes de estudio	80
G	6	Control de becas	90
G	7	Control de calificaciones	70
M	8	Becas de admisiones	80
M	9	Inscripción y re-inscripción	80
M	10	Listas y actas	90
M	11	Corte bancario	80
M	12	Cobro de extraordinarios	80
M	13	Facturación Preparatoria	90
M	14	Facturación Universidad	90
CH	15	Reportes de aspirantes	70
CH	16	Creación de grupos	80
CH	17	Búsqueda de estudiantes	90
CH	18	Generación recibos pago	70
CH	19	Reportes de evaluaciones	80

Tabla 1010.Porcentaje de aceptación proceso tradicional

	#	Módulo PROSER	Porcentaje de aceptación
G	20	Estado de cuenta	90
G	21	Acuerdos y prorrogas	80
G	22	Examen de admisión	90
G	23	Presentación de examen en Web	90
G	24	Expediente tutorías	80
G	25	Plantilla Dossier (Web)	100
G	26	Expediente profesores	80
M	27	Bitácora	80
M	28	Reportes de pagos	100
M	29	Evaluación a profesores*	70
M	30	Autoevaluación de profesores	80
M	31	Control de documentos	90
M	32	Expediente de egresados/titulación	90
M	33	Expediente de Inglés	80
CH	34	Recepción de documentos	90
CH	35	Planeación de periodos	90
CH	36	Expediente clínico	90
CH	37	Expediente Servicio social	90
CH	38	Expediente Prácticas profesionales	90

Tabla 11. Porcentaje de aceptación PROSER

Es notable la aceptación de los módulos con PROSER. Las gráfica 10 muestra cómo fue la aceptación de los módulos durante la última revisión, para obtener esta gráfica se tomó en cuenta cada valor correspondiente en la tabla 10 y en la tabla 11, en el eje de las X se muestran las letras que corresponden en su caso a cada uno de los módulos desarrollados, siendo diferentes para cada proceso. En el eje Y se encuentra el porcentaje de aceptación.



**Gráfica 10. Aceptación tradicional Vs PROSER**

La media del porcentaje de aceptación en el proceso tradicional es de  $\bar{x}=80$ , mientras que con PROSER el promedio fue de  $\bar{x}=86$ .

Ahora bien, en la gráfica 11 se muestra una gráfica que compara el porcentaje de aceptación durante la primera revisión, esta gráfica es interesante, ya que desde la aceptación inicial del cliente puede saberse que el cliente estaba conforme con lo que se estaba desarrollando. La claridad en los requisitos del cliente es un prerequisite para obtener este grado de aceptación aun desde la primera revisión.



## V. DISCUSIÓN

Los resultados han confirmado la hipótesis y se ha cumplido con los objetivos que se plantearon en esta tesis, seguir un procedimiento con reglas establecidas desde el inicio junto con el cliente, conduce a un proceso ordenado y controlado.

La variable número de modificaciones comparó el número de modificaciones solicitadas por el cliente para cada módulo durante la primera y la última revisión, en la primera revisión se evidencia si el requerimiento había sido redactado correctamente y que ambas partes, desarrollador y usuario coincidían en cuáles eran con exactitud los requerimientos.

En la comparación de medias, la variable número de modificaciones, en la última revisión con el proceso tradicional existieron grandes diferencias evidenciando al PROSER como un proceso que con más revisiones disminuye las modificaciones en el día de la entrega debido a que requiere una revisión intermedia que resulta en claridad en la comprensión, seguimiento y el desarrollo de cada uno de los requerimientos con el PROSER.

Esto indica que el cliente percibe que su sistema está siendo desarrollado con calidad y apegado a sus requerimientos ya que las modificaciones son pocas y el entregable está más cerca de lo que solicitó.

Al disminuir las quejas y molestias de los usuarios, en los desarrolladores disminuyó la frustración y el estrés debido a las múltiples modificaciones que se le hacían al sistema, las modificaciones requieren tiempo de re-programación, este tiempo extra dedicado a las modificaciones puede ser utilizado para otros desarrollos o para correcta documentación del proceso.

Al destinar más tiempo a cada módulo que debió haber finalizado, retrasa esa entrega y todas las demás contempladas en el cronograma, generando un efecto en cadena, se tienen que volver a negociar las entregas, se corre el riesgo de culpar al equipo de desarrollo por un retraso cuando son numerosas las modificaciones que solicita el usuario, el equipo de desarrollo culpará al usuario por no expresarse bien o a la persona que le pasó el requerimiento, otro efecto en cadena.

Haciendo evidente la importancia de la documentación de los requerimientos y su validación junto al cliente antes de comenzar el desarrollo.

La entrega final con el proceso tradicional llega a tener incluso más de 2 meses de retraso, mientras que en el proceso simple de elicitation de requerimientos PROSER sólo hay máximo 15 días retraso, traduciendo en que el cliente perciba que la empresa es responsable con sus tiempos y le brinda la sensación de que se le da importancia a su necesidad.

Al realizar al menos dos revisiones antes de la entrega final el cliente percibe que se modifica hasta que queda satisfecho con su módulo y cuando no se establecen estas dos revisiones el cliente percibe que se le entrega algo que no pidió o que no se entendió lo que necesitaba, establecer al menos dos revisiones previas a la entrega final es buena táctica para depurar, reordenar, corregir y dar máxima calidad en un entregable de desarrollo de software.

El porcentaje de aceptación es importante para la medición de la satisfacción del cliente, ya que en este caso existía un precedente de sistema al cual el usuario ya estaba acostumbrado, conociendo los atajos para llegar a la información y conociendo al 100 por ciento sus alcances y limitaciones.

Al llegar un nuevo sistema y además con tecnologías más visuales y componentes más atractivos, el usuario puede tener la sensación de requerir un conocimiento más profundo de tecnologías de la información o que se está utilizando una herramienta no comprensible, por otro lado están los usuarios con apertura a los cambios quienes son entusiastas, ven en cada cambio una oportunidad de mejora y aprendizaje, participan activamente en la mejora del sistema.

El alcance de un sistema informático es tan ilimitado como los recursos que se le asignen, los usuarios tienen la oportunidad de mejorar sus procesos, cuando en su mayoría los trabajaban en Excel con muchos archivos y respaldos para prevenir la pérdida de información, los usuarios tienen libertad para ser creativos solicitando lo que necesitan para facilitar su trabajo y sus procesos en todo momento.

Para la variable porcentaje de aceptación se solicitó al desarrollador que al entregar cada módulo calificara en una escala de 0 a 100 la actitud que haya mostrado el usuario con su entrega,



este indicador va de la mano con la cantidad de modificaciones que solicitan en la entrega, ya que, si hace muchas solicitudes de cambios sobre un entregable significa que hubo poca aceptación.

La ventaja de trabajar esta investigación en una empresa pequeña es que se proporcionó la libertad y la confianza para aplicar el procedimiento, además de la libertad para comunicarle al cliente lo que se estaba realizando y junto con los desarrolladores poder realizar la documentación de los requerimientos, gracias a todo esto fue posible recabar los resultados para esta investigación.

El proceso no termina con la entrega del módulo, debe tener continuidad con el cotejo de los manuales ante la Plantilla para la documentación de procesos (Anexo E), el usuario percibe como un trabajo terminado cuando se le ha entregado un manual y un manual está terminado hasta que ya no se solicitan modificaciones.

En la experiencia de la autora de esta investigación la realización del manual de usuario requiere de un lenguaje coloquial lo menos técnico posible, lo más concreto y directo en cuanto al uso de un sistema o módulo, los mejores manuales son los que enumeran los pasos a seguir para lograr realizar una actividad dentro del sistema, la plantilla para la documentación de procesos se diseñó con el propósito de que el manual de usuario que se menciona sea fácil de realizar pero se necesita hacer la prueba y obtener los resultados.

## VI. BIBLIOGRAFÍA

- Alarcón, R. (1996). Levantamiento de requerimientos y diseño de sistemas. Chile: Universidad de Concepción.
- Alonso, F., Martínez, L., y Segovia, J. (2005). Introducción a la Ingeniería del Software. España: Delta publicaciones universitarias.
- Barrish, J. (2014). The 20 Most popular Project Management Software Products. Disponible en: <http://blog.capterra.com/20-popular-project-management-software-products-infographic/>
- Campderrich, B. (2003). Ingeniería del software. Barcelona: UOC.
- Chemuturi, M. (2010). Mastering Software Quality Assurance: best practices, tools and techniques for software developers. USA.
- Choudhury, J. (2013). Challenges for a Technical Communicator in the Global World. Computing Now. Disponible en: <http://www.computer.org/portal/web/computingnow>
- Davis, A. (2005). Just Enough Requirements Management: where software development meets marketing. USA: Dorset House Publishing.
- Goguen, J., & Linde, C. (1993). Techniques for Requirements Elicitation. IEEE Computer Society, 152-164.
- IIE. (2003). Institute of Industrial Engineers, México. Consultado el 05 Junio de 2015 en el Institute of Industrial Engineers, México. Disponible en: <http://www.iie.org.mx/boletin032003/ind.pdf>
- Kleinman, K. (2001). ENIAC Programmers Project,. Consultado el 10 de Septiembre de 2014. Disponible en: <http://www.eniacprogrammers.org/index.shtml>.
- Lozano, R. (2013). Problemáticas actuales de las Pymes en México. Consultado el 19 de septiembre de 2013 en: SoyConta Innovación Contable. Problemáticas actuales de las Pymes en México. Disponible en: <http://www.soyconta.mx/problematicas-actuales-de-las-pymes-en-mexico/>
- Manies, M., y Uolevis, N. (2011). La elicitation de requerimientos en el contexto de un proyecto de software. Ing. USBMed, Vol.2 N°2.
- Mohapatra, P. (2010). Software Engineering. Daryagan, Delhi: New Age International.
- Oktaba, H. (2008). Historia y futuro de la ingeniería de software. Visión de Barry Boehm: Consultado el 10 de febrero del 2013. Disponible en: <http://sg.com.mx/content/view/356>

- Páez, N., Fontdevila, D., y Suarez, P. (2014). Construcción de software: una mirada ágil. Buenos Aires: EDUNTREF.
- Phillips, J. (2009). CBAP Certified Business Analysis Professional All-in-One Exam Guide. Disponible en: <http://www.books24x7.com>
- Preece, J., Rogers, Y., Sharp, Benyon, D., Holland, S. y Carey, T. (1994). Human computer interaction. Salt-Lake, UT: Addison-Wesley Longman.
- Pressman, R. (1997). “Ingeniería del Software: Un enfoque práctico”. (5 ed). México: McGrawHill.
- Pressman, R. (2005). Software engineering, A practitioner’s Approach (6ª ed). USA: McGraw-Hill
- Pressman, R. (2010). Software engineering. A practitioner’s Approach. (7ª ed). McGraw-Hill
- Promexico. (2013). Pymes, eslabón fundamental para el crecimiento en México. Obtenido de Promexico. Pymes, eslabón fundamental para el crecimiento en México, consultado el 19 de septiembre de 2013, disponible en: [//www.promexico.gob.mx/negocios-internacionales/pymes-eslabon-fundamental-para-el-crecimiento-en-mexico.html](http://www.promexico.gob.mx/negocios-internacionales/pymes-eslabon-fundamental-para-el-crecimiento-en-mexico.html)
- Raghavan , S., Zelesnik, G., y Ford, G. (1994). Lecture Notes on Requirements Elicitation. Educational Materials CMU/SEI-94-EM- 10, Software Engineering Institute, Carnegie Mellon University.
- Raghavan, S., Zelesnik , G., y Ford , G. (1994). Lecture Notes on Requirements Elicitation. . Obtenido de Educational Materials. Software Engineering Institute of Canergie Mellon University. Disponible en: [http://resources.sei.cmu.edu/asset\\_files/EducationalMaterial/1994\\_011\\_001\\_16238.pdf](http://resources.sei.cmu.edu/asset_files/EducationalMaterial/1994_011_001_16238.pdf)
- Ramos, J. L. (Noviembre-Diciembre 1999 de 1998). “Las generaciones de computadoras”. Revista Ciencia y Desarrollo., 45-51.
- Schwaber, K., & Beedle, M. (2001). Agile software development with scrum. Nueva Jersey: Prentice Hall.
- Secretaría de Economía . (2015). Secretaría de Economía -Prosoft. Disponible en: PROSOFT 3.0: <http://www.prosoft.economia.gob.mx/acercade/>
- Secretaria de Economía. (2015). Secretaría de Economía. México. Disponible en: <http://www.economia.gob.mx/>
- Sommerville, I. (2005). Ingeniería de software. (7 ed.). Madrid: Pearson Educación.

Torres, Y., y Noda, Z. (2012). Propuesta para el levantamiento de requisitos funcionales .  
Madrid: Académica Española.

Young, R. (2006). Project Requirements A Guide to Best Practices. EUA: Management Concepts  
Inc.

Zapata, C., Giraldo, G., y Mesa, J. (2010). Una propuesta de metaontología para la educación de  
requisitos. *Ingeniare: Revista Chilena de Ingeniería*, 18(1), 26-37.

# **ANEXO A**

**IEEE Std. 830-1998**

**IEEE Std 830-1998**

(Revision of  
IEEE Std 830-1993)

# IEEE Recommended Practice for Software Requirements Specifications

Sponsor

**Software Engineering Standards Committee  
of the  
IEEE Computer Society**

Approved 25 June 1998

**IEEE-SA Standards Board**

**Abstract:** The content and qualities of a good software requirements specification (SRS) are described and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. Guidelines for compliance with IEEE/EIA 12207.1-1997 are also provided.

**Keywords:** contract, customer, prototyping, software requirements specification, supplier, system requirements specifications

---

The Institute of Electrical and Electronics Engineers, Inc.  
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1998 by the Institute of Electrical and Electronics Engineers, Inc.  
All rights reserved. Published 1998. Printed in the United States of America.

ISBN 0-7381-0332-2

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

**IEEE Standards** documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board  
445 Hoes Lane  
P.O. Box 1331  
Piscataway, NJ 08855-1331  
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (978) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

# Introduction

(This introduction is not a part of IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.)

This recommended practice describes recommended approaches for the specification of software requirements. It is based on a model in which the result of the software requirements specification process is an unambiguous and complete specification document. It should help

- a) Software customers to accurately describe what they wish to obtain;
- b) Software suppliers to understand exactly what the customer wants;
- c) Individuals to accomplish the following goals:
  - 1) Develop a standard software requirements specification (SRS) outline for their own organizations;
  - 2) Define the format and content of their specific software requirements specifications;
  - 3) Develop additional local supporting items such as an SRS quality checklist, or an SRS writer's handbook.

To the customers, suppliers, and other individuals, a good SRS should provide several specific benefits, such as the following:

- *Establish the basis for agreement between the customers and the suppliers on what the software product is to do.* The complete description of the functions to be performed by the software specified in the SRS will assist the potential users to determine if the software specified meets their needs or how the software must be modified to meet their needs.
- *Reduce the development effort.* The preparation of the SRS forces the various concerned groups in the customer's organization to consider rigorously all of the requirements before design begins and reduces later redesign, recoding, and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct.
- *Provide a basis for estimating costs and schedules.* The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates.
- *Provide a baseline for validation and verification.* Organizations can develop their validation and verification plans much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured.
- *Facilitate transfer.* The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organization, and suppliers find it easier to transfer it to new customers.
- *Serve as a basis for enhancement.* Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

The readers of this document are referred to Annex B for guidelines for using this recommended practice to meet the requirements of IEEE/EIA 12207.1-1997, IEEE/EIA Guide—Industry Implementation of ISO/IEC 12207: 1995, Standard for Information Technology—Software life cycle processes—Life cycle data.



## Participants

This recommended practice was prepared by the Life Cycle Data Harmonization Working Group of the Software Engineering Standards Committee of the IEEE Computer Society. At the time this recommended practice was approved, the working group consisted of the following members:

### Leonard L. Tripp, *Chair*

Edward Byrne  
Paul R. Croll  
Perry DeWeese  
Robin Fralick  
Marilyn Ginsberg-Finner  
John Harauz  
Mark Henley

Dennis Lawrence  
David Maibor  
Ray Milovanovic  
James Moore  
Timothy Niesen  
Dennis Rilling

Terry Rout  
Richard Schmidt  
Norman F. Schneidewind  
David Schultz  
Basil Sherlund  
Peter Voldner  
Ronald Wade

The following persons were on the balloting committee:

Syed Ali  
Theodore K. Atchinson  
Mikhail Auguston  
Robert E. Barry  
Leo Beltracchi  
H. Ronald Berlack  
Richard E. Biehl  
Michael A. Blackledge  
Sandro Bologna  
Juris Borzovs  
Kathleen L. Briggs  
M. Scott Buck  
Michael Caldwell  
James E. Cardow  
Enrico A. Carrara  
Lawrence Catchpole  
Keith Chan  
Antonio M. Cicu  
Theo Clarke  
Sylvain Clermont  
Rosemary Coleman  
Virgil Lee Cooper  
W. W. Geoff Cozens  
Paul R. Croll  
Gregory T. Daich  
Geoffrey Darnton  
Taz Daughtrey  
Bostjan K. Derganc  
Perry R. DeWeese  
James Do  
Evelyn S. Dow  
Carl Einar Dragstedt  
Sherman Eagles  
Christof Ebert  
Leo Egan  
Richard E. Fairley  
John W. Fendrich  
Jay Forster  
Kirby Fortenberry  
Eva Freund  
Richard C. Fries  
Roger U. Fujii  
Adel N. Ghannam  
Marilyn Ginsberg-Finner  
John Garth Glynn  
Julio Gonzalez-Sanz  
L. M. Gunther

David A. Gustafson  
Jon D. Hagar  
John Harauz  
Robert T. Harley  
Herbert Hecht  
William Hefley  
Manfred Hein  
Mark Heinrich  
Mark Henley  
Debra Herrmann  
John W. Horch  
Jerry Huller  
Peter L. Hung  
George Jackelen  
Frank V. Jorgensen  
William S. Junk  
George X. Kambic  
Richard Karcich  
Ron S. Kenett  
Judith S. Kerner  
Robert J. Kierzyk  
Dwayne L. Knirk  
Shaye Koenig  
Thomas M. Kurihara  
John B. Lane  
J. Dennis Lawrence  
Fang Ching Lim  
William M. Lively  
James J. Longbucco  
Dieter Look  
John Lord  
Stan Magee  
David Maibor  
Harold Mains  
Robert A. Martin  
Tomoo Matsubara  
Mike McAndrew  
Patrick D. McCray  
Christopher McMacken  
Jerome W. Mersky  
Bret Michael  
Alan Miller  
Celia H. Modell  
James W. Moore  
Pavol Navrat  
Myrna L. Olson

Indradeb P. Pal  
Alex Polack  
Peter T. Poon  
Lawrence S. Przybylski  
Kenneth R. Ptack  
Annette D. Reilly  
Dennis Rilling  
Andrew P. Sage  
Helmut Sandmayr  
Stephen R. Schach  
Hans Schaefer  
Norman Schneidewind  
David J. Schultz  
Lisa A. Selmon  
Robert W. Shillato  
David M. Siefert  
Carl A. Singer  
James M. Sivak  
Richard S. Sky  
Nancy M. Smith  
Melford E. Smyre  
Harry M. Sneed  
Alfred R. Sorkowitz  
Donald W. Sova  
Luca Spotorno  
Julia Stesney  
Fred J. Strauss  
Christine Brown Strysik  
Toru Takeshita  
Richard H. Thayer  
Booker Thomas  
Patricia Trellue  
Theodore J. Urbanowicz  
Glenn D. Venables  
Udo Voges  
David D. Walden  
Dolores Wallace  
William M. Walsh  
John W. Walz  
Camille SWhite-Partain  
Scott A. Whitmire  
P. A. Wolfgang  
Paul R. Work  
Natalie C. Yopconka  
Janusz Zalewski  
Geraldine Zimmerman  
Peter F. Zoll

When the IEEE-SA Standards Board approved this recommended practice on 25 June 1998, it had the following membership:

**Richard J. Holleman**, *Chair*

**Donald N. Heirman**, *Vice Chair*

**Judith Gorman**, *Secretary*

Satish K. Aggarwal  
Clyde R. Camp  
James T. Carlo  
Gary R. Engmann  
Harold E. Epstein  
Jay Forster\*  
Thomas F. Garrity  
Ruben D. Garzon

James H. Gurney  
Jim D. Isaak  
Lowell G. Johnson  
Robert Kennelly  
E. G. "Al" Kiener  
Joseph L. Koepfinger\*  
Stephen R. Lambert  
Jim Logothetis  
Donald C. Loughry

L. Bruce McClung  
Louis-François Pau  
Ronald C. Petersen  
Gerald H. Peterson  
John B. Posey  
Gary S. Robinson  
Hans E. Weinrich  
Donald W. Zipse

\*Member Emeritus

Valerie E. Zelenty  
*IEEE Standards Project Editor*

# Contents

1. Overview.....	1
1.1 Scope.....	1
2. References.....	2
3. Definitions.....	2
4. Considerations for producing a good SRS.....	3
4.1 Nature of the SRS .....	3
4.2 Environment of the SRS .....	3
4.3 Characteristics of a good SRS.....	4
4.4 Joint preparation of the SRS .....	8
4.5 SRS evolution .....	8
4.6 Prototyping.....	9
4.7 Embedding design in the SRS.....	9
4.8 Embedding project requirements in the SRS .....	10
5. The parts of an SRS .....	10
5.1 Introduction (Section 1 of the SRS).....	11
5.2 Overall description (Section 2 of the SRS).....	12
5.3 Specific requirements (Section 3 of the SRS).....	15
5.4 Supporting information .....	19
Annex A (informative) SRS templates.....	21
Annex B (informative) Guidelines for compliance with IEEE/EIA 12207.1-1997 .....	27

# IEEE Recommended Practice for Software Requirements Specifications

## 1. Overview

This recommended practice describes recommended approaches for the specification of software requirements. It is divided into five clauses. Clause 1 explains the scope of this recommended practice. Clause 2 lists the references made to other standards. Clause 3 provides definitions of specific terms used. Clause 4 provides background information for writing a good SRS. Clause 5 discusses each of the essential parts of an SRS. This recommended practice also has two annexes, one which provides alternate format templates, and one which provides guidelines for compliance with IEEE/EIA 12207.1-1997.

### 1.1 Scope

This is a recommended practice for writing software requirements specifications. It describes the content and qualities of a good software requirements specification (SRS) and presents several sample SRS outlines.

This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. However, application to already-developed software could be counterproductive.

When software is embedded in some larger system, such as medical equipment, then issues beyond those identified in this recommended practice may have to be addressed.

This recommended practice describes the process of creating a product and the content of the product. The product is an SRS. This recommended practice can be used to create such an SRS directly or can be used as a model for a more specific standard.

This recommended practice does not identify any specific method, nomenclature, or tool for preparing an SRS.

## 2. References

This recommended practice shall be used in conjunction with the following publications.

ASTM E1340-96, Standard Guide for Rapid Prototyping of Computerized Systems.<sup>1</sup>

IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.<sup>2</sup>

IEEE Std 730-1998, IEEE Standard for Software Quality Assurance Plans.

IEEE Std 730.1-1995, IEEE Guide for Software Quality Assurance Planning.

IEEE Std 828-1998, IEEE Standard for Software Configuration Management Plans.<sup>3</sup>

IEEE Std 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software.

IEEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software.

IEEE Std 1002-1987 (Reaff 1992), IEEE Standard Taxonomy for Software Engineering Standards.

IEEE Std 1012-1998, IEEE Standard for Software Verification and Validation.

IEEE Std 1012a-1998, IEEE Standard for Software Verification and Validation: Content Map to IEEE/EIA 12207.1-1997.<sup>4</sup>

IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions.<sup>5</sup>

IEEE Std 1028-1997, IEEE Standard for Software Reviews.

IEEE Std 1042-1987 (Reaff 1993), IEEE Guide to Software Configuration Management.

IEEE P1058/D2.1, Draft Standard for Software Project Management Plans, dated 5 August 1998.<sup>6</sup>

IEEE Std 1058a-1998, IEEE Standard for Software Project Management Plans: Content Map to IEEE/EIA 12207.1-1997.<sup>7</sup>

IEEE Std 1074-1997, IEEE Standard for Developing Software Life Cycle Processes.

IEEE Std 1233, 1998 Edition, IEEE Guide for Developing System Requirements Specifications.<sup>8</sup>

---

<sup>1</sup>ASTM publications are available from the American Society for Testing and Materials, 100 Barr Harbor Drive, West Conshohocken, PA 19428-2959, USA.

<sup>2</sup>IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

<sup>3</sup>As this standard goes to press, IEEE Std 828-1998; IEEE Std 1012a-1998; IEEE Std 1016-1998; and IEEE Std 1233, 1998 Edition are approved but not yet published. The draft standards are, however, available from the IEEE. Anticipated publication date is Fall 1998. Contact the IEEE Standards Department at 1 (732) 562-3800 for status information.

<sup>4</sup>See Footnote 3.

<sup>5</sup>See Footnote 3.

<sup>6</sup>Upon approval of IEEE P1058 by the IEEE-SA Standards Board, this standard will be integrated with IEEE Std 1058a-1998 and published as IEEE Std 1058, 1998 Edition. Approval is expected 8 December 1998.

<sup>7</sup>As this standard goes to press, IEEE Std 1058a-1998 is approved but not yet published. The draft standard is, however, available from the IEEE. Anticipated publication date is December 1998. Contact the IEEE Standards Department at 1 (732) 562-3800 for status information. See Footnote 6.

<sup>8</sup>See Footnote 3.

### 3. Definitions

In general the definitions of terms used in this recommended practice conform to the definitions provided in IEEE Std 610.12-1990. The definitions below are key terms as they are used in this recommended practice.

**3.1 contract:** A legally binding document agreed upon by the customer and supplier. This includes the technical and organizational requirements, cost, and schedule for a product. A contract may also contain informal but useful information such as the commitments or expectations of the parties involved.

**3.2 customer:** The person, or persons, who pay for the product and usually (but not necessarily) decide the requirements. In the context of this recommended practice the customer and the supplier may be members of the same organization.

**3.3 supplier:** The person, or persons, who produce a product for a customer. In the context of this recommended practice, the customer and the supplier may be members of the same organization.

**3.4 user:** The person, or persons, who operate or interact directly with the product. The user(s) and the customer(s) are often not the same person(s).

### 4. Considerations for producing a good SRS

This clause provides background information that should be considered when writing an SRS. This includes the following:

- a) Nature of the SRS;
- b) Environment of the SRS;
- c) Characteristics of a good SRS;
- d) Joint preparation of the SRS;
- e) SRS evolution;
- f) Prototyping;
- g) Embedding design in the SRS;
- h) Embedding project requirements in the SRS.

#### 4.1 Nature of the SRS

The SRS is a specification for a particular software product, program, or set of programs that performs certain functions in a specific environment. The SRS may be written by one or more representatives of the supplier, one or more representatives of the customer, or by both. Subclause 4.4 recommends both.

The basic issues that the SRS writer(s) shall address are the following:

- a) *Functionality.* What is the software supposed to do?
- b) *External interfaces.* How does the software interact with people, the system's hardware, other hardware, and other software?
- c) *Performance.* What is the speed, availability, response time, recovery time of various software functions, etc.?
- d) *Attributes.* What are the portability, correctness, maintainability, security, etc. considerations?
- e) *Design constraints imposed on an implementation.* Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

The SRS writer(s) should avoid placing either design or project requirements in the SRS.

For recommended contents of an SRS see Clause 5.

## 4.2 Environment of the SRS

It is important to consider the part that the SRS plays in the total project plan, which is defined in IEEE Std 610.12-1990. The software may contain essentially all the functionality of the project or it may be part of a larger system. In the latter case typically there will be an SRS that will state the interfaces between the system and its software portion, and will place external performance and functionality requirements upon the software portion. Of course the SRS should then agree with and expand upon these system requirements.

IEEE Std 1074-1997 describes the steps in the software life cycle and the applicable inputs for each step. Other standards, such as those listed in Clause 2, relate to other parts of the software life cycle and so may complement software requirements.

Since the SRS has a specific role to play in the software development process, the SRS writer(s) should be careful not to go beyond the bounds of that role. This means the SRS

- a) Should correctly define all of the software requirements. A software requirement may exist because of the nature of the task to be solved or because of a special characteristic of the project.
- b) Should not describe any design or implementation details. These should be described in the design stage of the project.
- c) Should not impose additional constraints on the software. These are properly specified in other documents such as a software quality assurance plan.

Therefore, a properly written SRS limits the range of valid designs, but does not specify any particular design.

## 4.3 Characteristics of a good SRS

An SRS should be

- a) Correct;
- b) Unambiguous;
- c) Complete;
- d) Consistent;
- e) Ranked for importance and/or stability;
- f) Verifiable;
- g) Modifiable;
- h) Traceable.

### 4.3.1 Correct

An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet.

There is no tool or procedure that ensures correctness. The SRS should be compared with any applicable superior specification, such as a system requirements specification, with other project documentation, and with other applicable standards, to ensure that it agrees. Alternatively the customer or user can determine if the SRS correctly reflects the actual needs. Traceability makes this procedure easier and less prone to error (see 4.3.8).

### 4.3.2 Unambiguous

An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term.

In cases where a term used in a particular context could have multiple meanings, the term should be included in a glossary where its meaning is made more specific.

An SRS is an important part of the requirements process of the software life cycle and is used in design, implementation, project monitoring, verification and validation, and in training as described in IEEE Std 1074-1997. The SRS should be unambiguous both to those who create it and to those who use it. However, these groups often do not have the same background and therefore do not tend to describe software requirements the same way. Representations that improve the requirements specification for the developer may be counterproductive in that they diminish understanding to the user and vice versa.

Subclauses 4.3.2.1 through 4.3.2.3 recommend how to avoid ambiguity.

#### **4.3.2.1 Natural language pitfalls**

Requirements are often written in natural language (e.g., English). Natural language is inherently ambiguous. A natural language SRS should be reviewed by an independent party to identify ambiguous use of language so that it can be corrected.

#### **4.3.2.2 Requirements specification languages**

One way to avoid the ambiguity inherent in natural language is to write the SRS in a particular requirements specification language. Its language processors automatically detect many lexical, syntactic, and semantic errors.

One disadvantage in the use of such languages is the length of time required to learn them. Also, many non-technical users find them unintelligible. Moreover, these languages tend to be better at expressing certain types of requirements and addressing certain types of systems. Thus, they may influence the requirements in subtle ways.

#### **4.3.2.3 Representation tools**

In general, requirements methods and languages and the tools that support them fall into three general categories—object, process, and behavioral. Object-oriented approaches organize the requirements in terms of real-world objects, their attributes, and the services performed by those objects. Process-based approaches organize the requirements into hierarchies of functions that communicate via data flows. Behavioral approaches describe external behavior of the system in terms of some abstract notion (such as predicate calculus), mathematical functions, or state machines.

The degree to which such tools and methods may be useful in preparing an SRS depends upon the size and complexity of the program. No attempt is made here to describe or endorse any particular tool.

When using any of these approaches it is best to retain the natural language descriptions. That way, customers unfamiliar with the notations can still understand the SRS.

### **4.3.3 Complete**

An SRS is complete if, and only if, it includes the following elements:

- a) All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.



- b) Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.
- c) Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.

#### **4.3.3.1 Use of TBDs**

Any SRS that uses the phrase “to be determined” (TBD) is not a complete SRS. The TBD is, however, occasionally necessary and should be accompanied by

- a) A description of the conditions causing the TBD (e.g., why an answer is not known) so that the situation can be resolved;
- b) A description of what must be done to eliminate the TBD, who is responsible for its elimination, and by when it must be eliminated.

#### **4.3.4 Consistent**

Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct (see 4.3.1).

##### **4.3.4.1 Internal consistency**

An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict. The three types of likely conflicts in an SRS are as follows:

- a) The specified characteristics of real-world objects may conflict. For example,
  - 1) The format of an output report may be described in one requirement as tabular but in another as textual.
  - 2) One requirement may state that all lights shall be green while another may state that all lights shall be blue.
- b) There may be logical or temporal conflict between two specified actions. For example,
  - 1) One requirement may specify that the program will add two inputs and another may specify that the program will multiply them.
  - 2) One requirement may state that “A” must always follow “B,” while another may require that “A and B” occur simultaneously.
- c) Two or more requirements may describe the same real-world object but use different terms for that object. For example, a program’s request for a user input may be called a “prompt” in one requirement and a “cue” in another. The use of standard terminology and definitions promotes consistency.

##### **4.3.5 Ranked for importance and/or stability**

An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement.

Typically, all of the requirements that relate to a software product are not equally important. Some requirements may be essential, especially for life-critical applications, while others may be desirable.

Each requirement in the SRS should be identified to make these differences clear and explicit. Identifying the requirements in the following manner helps:

- a) Have customers give more careful consideration to each requirement, which often clarifies any hidden assumptions they may have.
- b) Have developers make correct design decisions and devote appropriate levels of effort to the different parts of the software product.

#### 4.3.5.1 Degree of stability

One method of identifying requirements uses the dimension of stability. Stability can be expressed in terms of the number of expected changes to any requirement based on experience or knowledge of forthcoming events that affect the organization, functions, and people supported by the software system.

#### 4.3.5.2 Degree of necessity

Another way to rank requirements is to distinguish classes of requirements as essential, conditional, and optional.

- a) *Essential*. Implies that the software will not be acceptable unless these requirements are provided in an agreed manner.
- b) *Conditional*. Implies that these are requirements that would enhance the software product, but would not make it unacceptable if they are absent.
- c) *Optional*. Implies a class of functions that may or may not be worthwhile. This gives the supplier the opportunity to propose something that exceeds the SRS.

#### 4.3.6 Verifiable

An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable.

Nonverifiable requirements include statements such as “works well,” “good human interface,” and “shall usually happen.” These requirements cannot be verified because it is impossible to define the terms “good,” “well,” or “usually.” The statement that “the program shall never enter an infinite loop” is nonverifiable because the testing of this quality is theoretically impossible.

An example of a verifiable statement is

*Output of the program shall be produced within 20 s of event  $\times$  60% of the time; and shall be produced within 30 s of event  $\times$  100% of the time.*

This statement can be verified because it uses concrete terms and measurable quantities.

If a method cannot be devised to determine whether the software meets a particular requirement, then that requirement should be removed or revised.

#### 4.3.7 Modifiable

An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an SRS to

- a) Have a coherent and easy-to-use organization with a table of contents, an index, and explicit cross-referencing;
- b) Not be redundant (i.e., the same requirement should not appear in more than one place in the SRS);
- c) Express each requirement separately, rather than intermixed with other requirements.

Redundancy itself is not an error, but it can easily lead to errors. Redundancy can occasionally help to make an SRS more readable, but a problem can arise when the redundant document is updated. For instance, a requirement may be altered in only one of the places where it appears. The SRS then becomes inconsistent. Whenever redundancy is necessary, the SRS should include explicit cross-references to make it modifiable.

#### 4.3.8 Traceable

An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. The following two types of traceability are recommended:

- a) *Backward traceability (i.e., to previous stages of development)*. This depends upon each requirement explicitly referencing its source in earlier documents.
- b) *Forward traceability (i.e., to all documents spawned by the SRS)*. This depends upon each requirement in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially important when the software product enters the operation and maintenance phase. As code and design documents are modified, it is essential to be able to ascertain the complete set of requirements that may be affected by those modifications.

### 4.4 Joint preparation of the SRS

The software development process should begin with supplier and customer agreement on what the completed software must do. This agreement, in the form of an SRS, should be jointly prepared. This is important because usually neither the customer nor the supplier is qualified to write a good SRS alone.

- a) Customers usually do not understand the software design and development process well enough to write a usable SRS.
- b) Suppliers usually do not understand the customer's problem and field of endeavor well enough to specify requirements for a satisfactory system.

Therefore, the customer and the supplier should work together to produce a well-written and completely understood SRS.

A special situation exists when a system and its software are both being defined concurrently. Then the functionality, interfaces, performance, and other attributes and constraints of the software are not predefined, but rather are jointly defined and subject to negotiation and change. This makes it more difficult, but no less important, to meet the characteristics stated in 4.3. In particular, an SRS that does not comply with the requirements of its parent system specification is incorrect.

This recommended practice does not specifically discuss style, language usage, or techniques of good writing. It is quite important, however, that an SRS be well written. General technical writing books can be used for guidance.

#### 4.5 SRS evolution

The SRS may need to evolve as the development of the software product progresses. It may be impossible to specify some details at the time the project is initiated (e.g., it may be impossible to define all of the screen formats for an interactive program during the requirements phase). Additional changes may ensue as deficiencies, shortcomings, and inaccuracies are discovered in the SRS.

Two major considerations in this process are the following:

- a) Requirements should be specified as completely and thoroughly as is known at the time, even if evolutionary revisions can be foreseen as inevitable. The fact that they are incomplete should be noted.
- b) A formal change process should be initiated to identify, control, track, and report projected changes. Approved changes in requirements should be incorporated in the SRS in such a way as to
  - 1) Provide an accurate and complete audit trail of changes;
  - 2) Permit the review of current and superseded portions of the SRS.

#### 4.6 Prototyping

Prototyping is used frequently during the requirements portion of a project. Many tools exist that allow a prototype, exhibiting some characteristics of a system, to be created very quickly and easily. See also ASTM E1340-96.

Prototypes are useful for the following reasons:

- a) The customer may be more likely to view the prototype and react to it than to read the SRS and react to it. Thus, the prototype provides quick feedback.
- b) The prototype displays unanticipated aspects of the systems behavior. Thus, it produces not only answers but also new questions. This helps reach closure on the SRS.
- c) An SRS based on a prototype tends to undergo less change during development, thus shortening development time.

A prototype should be used as a way to elicit software requirements. Some characteristics such as screen or report formats can be extracted directly from the prototype. Other requirements can be inferred by running experiments with the prototype.

#### 4.7 Embedding design in the SRS

A requirement specifies an externally visible function or attribute of a system. A design describes a particular subcomponent of a system and/or its interfaces with other subcomponents. The SRS writer(s) should clearly distinguish between identifying required design constraints and projecting a specific design. Note that every requirement in the SRS limits design alternatives. This does not mean, though, that every requirement is design.

The SRS should specify what functions are to be performed on what data to produce what results at what location for whom. The SRS should focus on the services to be performed. The SRS should not normally specify design items such as the following:

- a) Partitioning the software into modules;
- b) Allocating functions to the modules;
- c) Describing the flow of information or control between modules;
- d) Choosing data structures.

#### **4.7.1 Necessary design requirements**

In special cases some requirements may severely restrict the design. For example, security or safety requirements may reflect directly into design such as the need to

- a) Keep certain functions in separate modules;
- b) Permit only limited communication between some areas of the program;
- c) Check data integrity for critical variables.

Examples of valid design constraints are physical requirements, performance requirements, software development standards, and software quality assurance standards.

Therefore, the requirements should be stated from a purely external viewpoint. When using models to illustrate the requirements, remember that the model only indicates the external behavior, and does not specify a design.

#### **4.8 Embedding project requirements in the SRS**

The SRS should address the software product, not the process of producing the software product.

Project requirements represent an understanding between the customer and the supplier about contractual matters pertaining to production of software and thus should not be included in the SRS. These normally include items such as

- a) Cost;
- b) Delivery schedules;
- c) Reporting procedures;
- d) Software development methods;
- e) Quality assurance;
- f) Validation and verification criteria;
- g) Acceptance procedures.

Project requirements are specified in other documents, typically in a software development plan, a software quality assurance plan, or a statement of work.

### **5. The parts of an SRS**

This clause discusses each of the essential parts of the SRS. These parts are arranged in Figure 1 in an outline that can serve as an example for writing an SRS.

While an SRS does not have to follow this outline or use the names given here for its parts, a good SRS should include all the information discussed here.

<b>Table of Contents</b>	
1. Introduction	
1.1 Purpose	
1.2 Scope	
1.3 Definitions, acronyms, and abbreviations	
1.4 References	
1.5 Overview	
2. Overall description	
2.1 Product perspective	
2.2 Product functions	
2.3 User characteristics	
2.4 Constraints	
2.5 Assumptions and dependencies	
3. Specific requirements (See 5.3.1 through 5.3.8 for explanations of possible specific requirements. See also Annex A for several different ways of organizing this section of the SRS.)	
Appendixes	
Index	

**Figure 1—Prototype SRS outline**

## **5.1 Introduction (Section 1 of the SRS)**

The introduction of the SRS should provide an overview of the entire SRS. It should contain the following subsections:

- a) Purpose;
- b) Scope;
- c) Definitions, acronyms, and abbreviations;
- d) References;
- e) Overview.

### **5.1.1 Purpose (1.1 of the SRS)**

This subsection should

- a) Delineate the purpose of the SRS;
- b) Specify the intended audience for the SRS.

### **5.1.2 Scope (1.2 of the SRS)**

This subsection should

- a) Identify the software product(s) to be produced by name (e.g., Host DBMS, Report Generator, etc.);
- b) Explain what the software product(s) will, and, if necessary, will not do;
- c) Describe the application of the software being specified, including relevant benefits, objectives, and goals;
- d) Be consistent with similar statements in higher-level specifications (e.g., the system requirements specification), if they exist.

### **5.1.3 Definitions, acronyms, and abbreviations (1.3 of the SRS)**

This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.

### **5.1.4 References (1.4 of the SRS)**

This subsection should

- a) Provide a complete list of all documents referenced elsewhere in the SRS;
- b) Identify each document by title, report number (if applicable), date, and publishing organization;
- c) Specify the sources from which the references can be obtained.

This information may be provided by reference to an appendix or to another document.

### **5.1.5 Overview (1.5 of the SRS)**

This subsection should

- a) Describe what the rest of the SRS contains;
- b) Explain how the SRS is organized.

## **5.2 Overall description (Section 2 of the SRS)**

This section of the SRS should describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3 of the SRS, and makes them easier to understand.

This section usually consists of six subsections, as follows:

- a) Product perspective;
- b) Product functions;
- c) User characteristics;
- d) Constraints;
- e) Assumptions and dependencies;
- f) Apportioning of requirements.

### **5.2.1 Product perspective (2.1 of the SRS)**

This subsection of the SRS should put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection should relate the requirements of that larger system to functionality of the software and should identify interfaces between that system and the software.

A block diagram showing the major components of the larger system, interconnections, and external interfaces can be helpful.

This subsection should also describe how the software operates inside various constraints. For example, these constraints could include

- a) System interfaces;
- b) User interfaces;
- c) Hardware interfaces;
- d) Software interfaces;
- e) Communications interfaces;
- f) Memory;
- g) Operations;
- h) Site adaptation requirements.

#### **5.2.1.1 System interfaces**

This should list each system interface and identify the functionality of the software to accomplish the system requirement and the interface description to match the system.

#### **5.2.1.2 User interfaces**

This should specify the following:

- a) *The logical characteristics of each interface between the software product and its users.* This includes those configuration characteristics (e.g., required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys) necessary to accomplish the software requirements.
- b) *All the aspects of optimizing the interface with the person who must use the system.* This may simply comprise a list of do's and don'ts on how the system will appear to the user. One example may be a requirement for the option of long or short error messages. Like all others, these requirements should be verifiable, e.g., "a clerk typist grade 4 can do function *X* in *Z* min after 1 h of training" rather than "a typist can do function *X*." (This may also be specified in the Software System Attributes under a section titled Ease of Use.)

#### **5.2.1.3 Hardware interfaces**

This should specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. For example, terminal support may specify full-screen support as opposed to line-by-line support.

#### **5.2.1.4 Software interfaces**

This should specify the use of other required software products (e.g., a data management system, an operating system, or a mathematical package), and interfaces with other application systems (e.g., the linkage between an accounts receivable system and a general ledger system). For each required software product, the following should be provided:

- Name;
- Mnemonic;
- Specification number;
- Version number;
- Source.



For each interface, the following should be provided:

- Discussion of the purpose of the interfacing software as related to this software product.
- Definition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document defining the interface is required.

#### **5.2.1.5 Communications interfaces**

This should specify the various interfaces to communications such as local network protocols, etc.

#### **5.2.1.6 Memory constraints**

This should specify any applicable characteristics and limits on primary and secondary memory.

#### **5.2.1.7 Operations**

This should specify the normal and special operations required by the user such as

- a) The various modes of operations in the user organization (e.g., user-initiated operations);
- b) Periods of interactive operations and periods of unattended operations;
- c) Data processing support functions;
- d) Backup and recovery operations.

NOTE—This is sometimes specified as part of the User Interfaces section.

#### **5.2.1.8 Site adaptation requirements**

This should

- a) Define the requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode (e.g., grid values, safety limits, etc.);
- b) Specify the site or mission-related features that should be modified to adapt the software to a particular installation.

### **5.2.2 Product functions (2.2 of the SRS)**

This subsection of the SRS should provide a summary of the major functions that the software will perform. For example, an SRS for an accounting program may use this part to address customer account maintenance, customer statement, and invoice preparation without mentioning the vast amount of detail that each of those functions requires.

Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product. Note that for the sake of clarity

- a) The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time.
- b) Textual or graphical methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product, but simply shows the logical relationships among variables.

### 5.2.3 User characteristics (2.3 of the SRS)

This subsection of the SRS should describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. It should not be used to state specific requirements, but rather should provide the reasons why certain specific requirements are later specified in Section 3 of the SRS.

### 5.2.4 Constraints (2.4 of the SRS)

This subsection of the SRS should provide a general description of any other items that will limit the developer's options. These include

- a) Regulatory policies;
- b) Hardware limitations (e.g., signal timing requirements);
- c) Interfaces to other applications;
- d) Parallel operation;
- e) Audit functions;
- f) Control functions;
- g) Higher-order language requirements;
- h) Signal handshake protocols (e.g., XON-XOFF, ACK-NACK);
- i) Reliability requirements;
- j) Criticality of the application;
- k) Safety and security considerations.

### 5.2.5 Assumptions and dependencies (2.5 of the SRS)

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption may be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.

### 5.2.6 Apportioning of requirements (2.6 of the SRS)

This subsection of the SRS should identify requirements that may be delayed until future versions of the system.

## 5.3 Specific requirements (Section 3 of the SRS)

This section of the SRS should contain all of the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input or in support of an output. As this is often the largest and most important part of the SRS, the following principles apply:

- a) Specific requirements should be stated in conformance with all the characteristics described in 4.3.
- b) Specific requirements should be cross-referenced to earlier documents that relate.
- c) All requirements should be uniquely identifiable.
- d) Careful attention should be given to organizing the requirements to maximize readability.

Before examining specific ways of organizing the requirements it is helpful to understand the various items that comprise requirements as described in 5.3.1 through 5.3.7.

### 5.3.1 External interfaces

This should be a detailed description of all inputs into and outputs from the software system. It should complement the interface descriptions in 5.2 and should not repeat information there.

It should include both content and format as follows:

- a) Name of item;
- b) Description of purpose;
- c) Source of input or destination of output;
- d) Valid range, accuracy, and/or tolerance;
- e) Units of measure;
- f) Timing;
- g) Relationships to other inputs/outputs;
- h) Screen formats/organization;
- i) Window formats/organization;
- j) Data formats;
- k) Command formats;
- l) End messages.

### 5.3.2 Functions

Functional requirements should define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as “shall” statements starting with “The system shall...”

These include

- a) Validity checks on the inputs
- b) Exact sequence of operations
- c) Responses to abnormal situations, including
  - 1) Overflow
  - 2) Communication facilities
  - 3) Error handling and recovery
- d) Effect of parameters
- e) Relationship of outputs to inputs, including
  - 1) Input/output sequences
  - 2) Formulas for input to output conversion

It may be appropriate to partition the functional requirements into subfunctions or subprocesses. This does not imply that the software design will also be partitioned that way.

### 5.3.3 Performance requirements

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on human interaction with the software as a whole. Static numerical requirements may include the following:

- a) The number of terminals to be supported;
- b) The number of simultaneous users to be supported;
- c) Amount and type of information to be handled.

Static numerical requirements are sometimes identified under a separate section entitled Capacity.

Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.

All of these requirements should be stated in measurable terms.

For example,

*95% of the transactions shall be processed in less than 1 s.*

rather than,

*An operator shall not have to wait for the transaction to complete.*

NOTE—Numerical limits applied to one specific function are normally specified as part of the processing subparagraph description of that function.

### **5.3.4 Logical database requirements**

This should specify the logical requirements for any information that is to be placed into a database. This may include the following:

- a) Types of information used by various functions;
- b) Frequency of use;
- c) Accessing capabilities;
- d) Data entities and their relationships;
- e) Integrity constraints;
- f) Data retention requirements.

### **5.3.5 Design constraints**

This should specify design constraints that can be imposed by other standards, hardware limitations, etc.

#### **5.3.5.1 Standards compliance**

This subsection should specify the requirements derived from existing standards or regulations. They may include the following:

- a) Report format;
- b) Data naming;
- c) Accounting procedures;
- d) Audit tracing.

For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.

### **5.3.6 Software system attributes**

There are a number of attributes of software that can serve as requirements. It is important that required attributes be specified so that their achievement can be objectively verified. Subclauses 5.3.6.1 through 5.3.6.5 provide a partial list of examples.

### **5.3.6.1 Reliability**

This should specify the factors required to establish the required reliability of the software system at time of delivery.

### **5.3.6.2 Availability**

This should specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart.

### **5.3.6.3 Security**

This should specify the factors that protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to

- a) Utilize certain cryptographical techniques;
- b) Keep specific log or history data sets;
- c) Assign certain functions to different modules;
- d) Restrict communications between some areas of the program;
- e) Check data integrity for critical variables.

### **5.3.6.4 Maintainability**

This should specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices.

### **5.3.6.5 Portability**

This should specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include the following:

- a) Percentage of components with host-dependent code;
- b) Percentage of code that is host dependent;
- c) Use of a proven portable language;
- d) Use of a particular compiler or language subset;
- e) Use of a particular operating system.

## **5.3.7 Organizing the specific requirements**

For anything but trivial systems the detailed requirements tend to be extensive. For this reason, it is recommended that careful consideration be given to organizing these in a manner optimal for understanding. There is no one optimal organization for all systems. Different classes of systems lend themselves to different organizations of requirements in Section 3 of the SRS. Some of these organizations are described in 5.3.7.1 through 5.3.7.7.

### **5.3.7.1 System mode**

Some systems behave quite differently depending on the mode of operation. For example, a control system may have different sets of functions depending on its mode: training, normal, or emergency. When organizing this section by mode, the outline in A.1 or A.2 should be used. The choice depends on whether interfaces and performance are dependent on mode.

### 5.3.7.2 User class

Some systems provide different sets of functions to different classes of users. For example, an elevator control system presents different capabilities to passengers, maintenance workers, and fire fighters. When organizing this section by user class, the outline in A.3 should be used.

### 5.3.7.3 Objects

Objects are real-world entities that have a counterpart within the system. For example, in a patient monitoring system, objects include patients, sensors, nurses, rooms, physicians, medicines, etc. Associated with each object is a set of attributes (of that object) and functions (performed by that object). These functions are also called services, methods, or processes. When organizing this section by object, the outline in A.4 should be used. Note that sets of objects may share attributes and services. These are grouped together as classes.

### 5.3.7.4 Feature

A feature is an externally desired service by the system that may require a sequence of inputs to effect the desired result. For example, in a telephone system, features include local call, call forwarding, and conference call. Each feature is generally described in a sequence of stimulus-response pairs. When organizing this section by feature, the outline in A.5 should be used.

### 5.3.7.5 Stimulus

Some systems can be best organized by describing their functions in terms of stimuli. For example, the functions of an automatic aircraft landing system may be organized into sections for loss of power, wind shear, sudden change in roll, vertical velocity excessive, etc. When organizing this section by stimulus, the outline in A.6 should be used.

### 5.3.7.6 Response

Some systems can be best organized by describing all the functions in support of the generation of a response. For example, the functions of a personnel system may be organized into sections corresponding to all functions associated with generating paychecks, all functions associated with generating a current list of employees, etc. The outline in A.6 (with all occurrences of stimulus replaced with response) should be used.

### 5.3.7.7 Functional hierarchy

When none of the above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by either common inputs, common outputs, or common internal data access. Data flow diagrams and data dictionaries can be used to show the relationships between and among the functions and data. When organizing this section by functional hierarchy, the outline in A.7 should be used.

### 5.3.8 Additional comments

Whenever a new SRS is contemplated, more than one of the organizational techniques given in 5.3.7.7 may be appropriate. In such cases, organize the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification. For example, see A.8 for an organization combining user class and feature. Any additional requirements may be put in a separate section at the end of the SRS.

There are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, finite state machines or state charts may prove helpful; when organizing by object, object-oriented

analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; and when organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.

In any of the outlines given in A.1 through A.8, those sections called “Functional Requirement *i*” may be described in native language (e.g., English), in pseudocode, in a system definition language, or in four subsections titled: Introduction, Inputs, Processing, and Outputs.

## 5.4 Supporting information

The supporting information makes the SRS easier to use. It includes the following:

- a) Table of contents;
- b) Index;
- c) Appendixes.

### 5.4.1 Table of contents and index

The table of contents and index are quite important and should follow general compositional practices.

### 5.4.2 Appendixes

The appendixes are not always considered part of the actual SRS and are not always necessary. They may include

- a) Sample input/output formats, descriptions of cost analysis studies, or results of user surveys;
- b) Supporting or background information that can help the readers of the SRS;
- c) A description of the problems to be solved by the software;
- d) Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements.

When appendixes are included, the SRS should explicitly state whether or not the appendixes are to be considered part of the requirements.

## Annex A

(informative)

### SRS templates

#### A.1 Template of SRS Section 3 organized by mode: Version 1

- 3. Specific requirements
  - 3.1 External interface requirements
    - 3.1.1 User interfaces
    - 3.1.2 Hardware interfaces
    - 3.1.3 Software interfaces
    - 3.1.4 Communications interfaces
  - 3.2 Functional requirements
    - 3.2.1 Mode 1
      - 3.2.1.1 Functional requirement 1.1
      - .
      - .
      - 3.2.1.*n* Functional requirement 1.*n*
    - 3.2.2 Mode 2
      - .
      - .
      - .
    - 3.2.*m* Mode *m*
      - 3.2.*m*.1 Functional requirement *m*.1
      - .
      - .
      - 3.2.*m*.*n* Functional requirement *m*.*n*
  - 3.3 Performance requirements
  - 3.4 Design constraints
  - 3.5 Software system attributes
  - 3.6 Other requirements

#### A.2 Template of SRS Section 3 organized by mode: Version 2

- 3. Specific requirements
  - 3.1. Functional requirements
    - 3.1.1 Mode 1
      - 3.1.1.1 External interfaces
        - 3.1.1.1.1 User interfaces
        - 3.1.1.1.2 Hardware interfaces
        - 3.1.1.1.3 Software interfaces
        - 3.1.1.1.4 Communications interfaces
      - 3.1.1.2 Functional requirements
        - 3.1.1.2.1 Functional requirement 1
        - .
        - .



- 3.1.1.2.*n* Functional requirement *n*
- 3.1.1.3 Performance
- 3.1.2 Mode 2
- .
- .
- .
- 3.1.*m* Mode *m*
- 3.2 Design constraints
- 3.3 Software system attributes
- 3.4 Other requirements

### **A.3 Template of SRS Section 3 organized by user class**

- 3. Specific requirements
  - 3.1 External interface requirements
    - 3.1.1 User interfaces
    - 3.1.2 Hardware interfaces
    - 3.1.3 Software interfaces
    - 3.1.4 Communications interfaces
  - 3.2 Functional requirements
    - 3.2.1 User class 1
      - 3.2.1.1 Functional requirement 1.1
      - .
      - .
      - 3.2.1.*n* Functional requirement 1.*n*
    - 3.2.2 User class 2
    - .
    - .
    - .
    - 3.2.*m* User class *m*
      - 3.2.*m*.1 Functional requirement *m*.1
      - .
      - .
      - 3.2.*m*.*n* Functional requirement *m*.*n*
  - 3.3 Performance requirements
  - 3.4 Design constraints
  - 3.5 Software system attributes
  - 3.6 Other requirements

### **A.4 Template of SRS Section 3 organized by object**

- 3. Specific requirements
  - 3.1 External interface requirements
    - 3.1.1 User interfaces
    - 3.1.2 Hardware interfaces
    - 3.1.3 Software interfaces
    - 3.1.4 Communications interfaces
  - 3.2 Classes/Objects
    - 3.2.1 Class/Object 1

- 3.2.1.1 Attributes (direct or inherited)
  - 3.2.1.1.1 Attribute 1
  - .
  - .
  - .
  - 3.2.1.1.*n* Attribute *n*
- 3.2.1.2 Functions (services, methods, direct or inherited)
  - 3.2.1.2.1 Functional requirement 1.1
  - .
  - .
  - .
  - 3.2.1.2.*m* Functional requirement 1.*m*
- 3.2.1.3 Messages (communications received or sent)
- 3.2.2 Class/Object 2
- .
- .
- .
- 3.2.*p* Class/Object *p*
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

### A.5 Template of SRS Section 3 organized by feature

- 3. Specific requirements
  - 3.1 External interface requirements
    - 3.1.1 User interfaces
    - 3.1.2 Hardware interfaces
    - 3.1.3 Software interfaces
    - 3.1.4 Communications interfaces
  - 3.2 System features
    - 3.2.1 System Feature 1
      - 3.2.1.1 Introduction/Purpose of feature
      - 3.2.1.2 Stimulus/Response sequence
      - 3.2.1.3 Associated functional requirements
        - 3.2.1.3.1 Functional requirement 1
        - .
        - .
        - .
        - 3.2.1.3.*n* Functional requirement *n*
    - 3.2.2 System feature 2
    - .
    - .
    - .
    - 3.2.*m* System feature *m*
    - .
    - .
    - .
  - 3.3 Performance requirements
  - 3.4 Design constraints
  - 3.5 Software system attributes
  - 3.6 Other requirements

## A.6 Template of SRS Section 3 organized by stimulus

- 3. Specific requirements
  - 3.1 External interface requirements
    - 3.1.1 User interfaces
    - 3.1.2 Hardware interfaces
    - 3.1.3 Software interfaces
    - 3.1.4 Communications interfaces
  - 3.2 Functional requirements
    - 3.2.1 Stimulus 1
      - 3.2.1.1 Functional requirement 1.1
      - .
      - .
      - 3.2.1.*n* Functional requirement 1.*n*
    - 3.2.2 Stimulus 2
    - .
    - .
    - 3.2.*m* Stimulus *m*
      - 3.2.*m*.1 Functional requirement *m*.1
      - .
      - .
      - 3.2.*m*.*n* Functional requirement *m*.*n*
  - 3.3 Performance requirements
  - 3.4 Design constraints
  - 3.5 Software system attributes
  - 3.6 Other requirements

## A.7 Template of SRS Section 3 organized by functional hierarchy

- 3. Specific requirements
  - 3.1 External interface requirements
    - 3.1.1 User interfaces
    - 3.1.2 Hardware interfaces
    - 3.1.3 Software interfaces
    - 3.1.4 Communications interfaces
  - 3.2 Functional requirements
    - 3.2.1 Information flows
      - 3.2.1.1 Data flow diagram 1
        - 3.2.1.1.1 Data entities
        - 3.2.1.1.2 Pertinent processes
        - 3.2.1.1.3 Topology
      - 3.2.1.2 Data flow diagram 2
        - 3.2.1.2.1 Data entities
        - 3.2.1.2.2 Pertinent processes
        - 3.2.1.2.3 Topology
      - .
      - .
      - 3.2.1.*n* Data flow diagram *n*

- 3.2.1.n.1 Data entities
- 3.2.1.n.2 Pertinent processes
- 3.2.1.n.3 Topology
- 3.2.2 Process descriptions
  - 3.2.2.1 Process 1
    - 3.2.2.1.1 Input data entities
    - 3.2.2.1.2 Algorithm or formula of process
    - 3.2.2.1.3 Affected data entities
  - 3.2.2.2 Process 2
    - 3.2.2.2.1 Input data entities
    - 3.2.2.2.2 Algorithm or formula of process
    - 3.2.2.2.3 Affected data entities
  - .
  - .
  - .
  - 3.2.2.m Process *m*
    - 3.2.2.m.1 Input data entities
    - 3.2.2.m.2 Algorithm or formula of process
    - 3.2.2.m.3 Affected data entities
- 3.2.3 Data construct specifications
  - 3.2.3.1 Construct 1
    - 3.2.3.1.1 Record type
    - 3.2.3.1.2 Constituent fields
  - 3.2.3.2 Construct 2
    - 3.2.3.2.1 Record type
    - 3.2.3.2.2 Constituent fields
  - .
  - .
  - .
  - 3.2.3.p Construct *p*
    - 3.2.3.p.1 Record type
    - 3.2.3.p.2 Constituent fields
- 3.2.4 Data dictionary
  - 3.2.4.1 Data element 1
    - 3.2.4.1.1 Name
    - 3.2.4.1.2 Representation
    - 3.2.4.1.3 Units/Format
    - 3.2.4.1.4 Precision/Accuracy
    - 3.2.4.1.5 Range
  - 3.2.4.2 Data element 2
    - 3.2.4.2.1 Name
    - 3.2.4.2.2 Representation
    - 3.2.4.2.3 Units/Format
    - 3.2.4.2.4 Precision/Accuracy
    - 3.2.4.2.5 Range
  - .
  - .
  - .
  - 3.2.4.q Data element *q*
    - 3.2.4.q.1 Name
    - 3.2.4.q.2 Representation
    - 3.2.4.q.3 Units/Format
    - 3.2.4.q.4 Precision/Accuracy
    - 3.2.4.q.5 Range

- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

## A.8 Template of SRS Section 3 showing multiple organizations

- 3. Specific requirements
  - 3.1 External interface requirements
    - 3.1.1 User interfaces
    - 3.1.2 Hardware interfaces
    - 3.1.3 Software interfaces
    - 3.1.4 Communications interfaces
  - 3.2 Functional requirements
    - 3.2.1 User class 1
      - 3.2.1.1 Feature 1.1
        - 3.2.1.1.1 Introduction/Purpose of feature
        - 3.2.1.1.2 Stimulus/Response sequence
        - 3.2.1.1.3 Associated functional requirements
      - 3.2.1.2 Feature 1.2
        - 3.2.1.2.1 Introduction/Purpose of feature
        - 3.2.1.2.2 Stimulus/Response sequence
        - 3.2.1.2.3 Associated functional requirements
      - .
      - .
      - .
      - 3.2.1.*m* Feature 1.*m*
        - 3.2.1.*m*.1 Introduction/Purpose of feature
        - 3.2.1.*m*.2 Stimulus/Response sequence
        - 3.2.1.*m*.3 Associated functional requirements
    - 3.2.2 User class 2
      - .
      - .
      - .
    - 3.2.*n* User class *n*
      - .
      - .
      - .
  - 3.3 Performance requirements
  - 3.4 Design constraints
  - 3.5 Software system attributes
  - 3.6 Other requirements

## Annex B

(informative)

### Guidelines for compliance with IEEE/EIA 12207.1-1997

#### B.1 Overview

The Software Engineering Standards Committee (SESC) of the IEEE Computer Society has endorsed the policy of adopting international standards. In 1995, the international standard, ISO/IEC 12207, Information technology—Software life cycle processes, was completed. The standard establishes a common framework for software life cycle processes, with well-defined terminology, that can be referenced by the software industry.

In 1995 the SESC evaluated ISO/IEC 12207 and decided that the standard should be adopted and serve as the basis for life cycle processes within the IEEE Software Engineering Collection. The IEEE adaptation of ISO/IEC 12207 is IEEE/EIA 12207.0-1996. It contains ISO/IEC 12207 and the following additions: improved compliance approach, life cycle process objectives, life cycle data objectives, and errata.

The implementation of ISO/IEC 12207 within the IEEE also includes the following:

- IEEE/EIA 12207.1-1997, IEEE/EIA Guide for Information Technology—Software life cycle processes—Life cycle data;
- IEEE/EIA 12207.2-1997, IEEE/EIA Guide for Information Technology—Software life cycle processes—Implementation considerations; and
- Additions to 11 SESC standards (i.e., IEEE Stds 730, 828, 829, 830, 1012, 1016, 1058, 1062, 1219, 1233, 1362) to define the correlation between the data produced by existing SESC standards and the data produced by the application of IEEE/EIA 12207.1-1997.

NOTE—Although IEEE/EIA 12207.1-1997 is a guide, it also contains provisions for application as a standard with specific compliance requirements. This annex treats 12207.1-1997 as a standard.

#### B.1.1 Scope and purpose

Both IEEE Std 830-1998 and IEEE/EIA 12207.1-1997 place requirements on a Software Requirements Description Document. The purpose of this annex is to explain the relationship between the two sets of requirements so that users producing documents intended to comply with both standards may do so.

#### B.2 Correlation

This clause explains the relationship between IEEE Std 830-1998 and IEEE/EIA 12207.0-1996 and IEEE/EIA 12207.1-1997 in the following areas: terminology, process, and life cycle data.

##### B.2.1 Terminology correlation

Both this recommended practice and IEEE/EIA 12207.0-1996 have similar semantics for the key terms of software, requirements, specification, supplier, developer, and maintainer. This recommended practice uses

the term “customer” where IEEE/EIA 12207.0-1996 uses “acquirer,” and this recommended practice uses “user” where IEEE/EIA 12207.0-1996 uses “operator.”

### B.2.2 Process correlation

IEEE/EIA 12207.0-1996 uses a process-oriented approach for describing the definition of a set of requirements for software. This recommended practice uses a product-oriented approach, where the product is a Software Requirements Description (SRD). There are natural process steps, namely the steps to create each portion of the SRD. These may be correlated with the process requirements of IEEE/EIA 12207.0-1996. The difference is that this recommended practice is focused on the development of software requirements whereas IEEE/EIA 12207.0-1996 provides an overall life cycle view and mentions Software Requirements Analysis as part of its Development Process. This recommended practice provides a greater level of detail on what is involved in the preparation of an SRD.

### B.2.3 Life cycle data correlation

IEEE/EIA 12207.0-1996 takes the viewpoint that the software requirements are derived from the system requirements. Therefore, it uses the term, “description” rather than “specification” to describe the software requirements. In a system in which software is a component, each requiring its own specification, there would be a System Requirements Specification (SRS) and one or more SRDs. If the term Software Requirements Specification had been used, there would be a confusion between an SRS referring to the system or software requirements. In the case where there is a stand-alone software system, IEEE/EIA 12207.1-1997 states “If the software is a stand-alone system, then this document should be a specification.”

## B.3 Content mapping

This clause provides details bearing on a claim that an SRS complying with this recommended practice would also achieve “document compliance” with the SRD described in IEEE/EIA 12207.1-1997. The requirements for document compliance are summarized in a single row of Table 1 of IEEE/EIA 12207.1-1997. That row is reproduced in Table B.1 of this recommended practice.

**Table B.1—Summary of requirements for an SRD  
excerpted from Table 1 of IEEE/EIA 12207.1-1997**

Information item	IEEE/EIA 12207.0-1996 Clause	Kind	IEEE/EIA 12207.1-1997 Clause	References
Software Requirements Description	5.1.1.4, 5.3.4.1, 5.3.4.2	Description (See note for 6.22.1 of IEEE/EIA 12207.1-1997.)	6.22	IEEE Std 830-1998; EIA/IEEE J-STD-016, F.2.3, F.2.4; MIL-STD 961D. Also see ISO/IEC 5806, 5807, 6593, 8631, 8790, and 11411 for guidance on use of notations.

The requirements for document compliance are discussed in the following subclauses:

- B.3.1 discusses compliance with the information requirements noted in column 2 of Table B.1 as prescribed by 5.1.1.4, 5.3.4.1, and 5.3.4.2 of IEEE/EIA 12207.0-1996.

- B.3.2 discusses compliance with the generic content guideline (the “kind” of document) noted in column 3 of Table B.1 as a “description”. The generic content guidelines for a “description” appear in 5.1 of IEEE/EIA 12207.1-1997.
- B.3.3 discusses compliance with the specific requirements for a Software Requirements Description noted in column 4 of Table B.1 as prescribed by 6.22 of IEEE/EIA 12207.1-1997.
- B.3.4 discusses compliance with the life cycle data objectives of Annex H of IEEE/EIA 12207.0-1996 as described in 4.2 of IEEE/EIA 12207.1-1997.

### B.3.1 Compliance with information requirements of IEEE/EIA 12207.0-1996

The information requirements for an SRD are those prescribed by 5.1.1.4, 5.3.4.1, and 5.3.4.2 of IEEE/EIA 12207.0-1996. The requirements are substantively identical to those considered in B.3.3 of this recommended practice.

### B.3.2 Compliance with generic content guidelines of IEEE/EIA 12207.1-1997

According to IEEE/EIA 12207.1-1997, the generic content guideline for an SRD is generally a description, as prescribed by 5.1 of IEEE/EIA 12207.1-1997. A complying description shall achieve the purpose stated in 5.1.1 and include the information listed in 5.1.2 of IEEE/EIA 12207.1-1997.

The purpose of a description is:

IEEE/EIA 12207.1-1997, subclause 5.1.1: Purpose: Describe a planned or actual function, design, performance, or process.

An SRD complying with this recommended practice would achieve the stated purpose.

Any description or specification complying with IEEE/EIA 12207.1-1997 shall satisfy the generic content requirements provided in 5.1.2 of that standard. Table B.2 of this recommended practice lists the generic content items and, where appropriate, references the clause of this recommended practice that requires the same information.

**Table B.2—Coverage of generic description requirements by IEEE Std 830-1998**

IEEE/EIA 12207.1-1997 generic content	Corresponding clauses of IEEE Std 830-1998	Additions to requirements of IEEE Std 830-1998
a) Date of issue and status	—	Date of issue and status shall be provided.
b) Scope	5.1.1 Scope	—
c) Issuing organization	—	Issuing organization shall be identified.
d) References	5.1.4 References	—
e) Context	5.1.2 Scope	—
f) Notation for description	4.3 Characteristics of a good SRS	—
g) Body	5. The parts of an SRS	—
h) Summary	5.1.1. Overview	—
i) Glossary	5.1.3 Definitions	—
j) Change history	—	Change history for the SRD shall be provided or referenced.



### B.3.3 Compliance with specific content requirements of IEEE/EIA 12207.1-1997

The specific content requirements for an SRD in IEEE/EIA 12207.1-1997 are prescribed by 6.22 of IEEE/EIA 12207.1-1997. A compliant SRD shall achieve the purpose stated in 6.22.1 of IEEE/EIA 12207.1-1997.

The purpose of the SRD is:

IEEE/EIA 12207.1-1997, subclause 6.22.1: Purpose: Specify the requirements for a software item and the methods to be used to ensure that each requirement has been met. Used as the basis for design and qualification testing of a software item.

An SRS complying with this recommended practice and meeting the additional requirements of Table B.3 of this recommended practice would achieve the stated purpose.

An SRD compliant with IEEE/EIA 12207.1-1997 shall satisfy the specific content requirements provided in 6.22.3 and 6.22.4 of that standard. Table B.3 of this recommended practice lists the specific content items and, where appropriate, references the clause of this recommended practice that requires the same information.

An SRD specified according the requirements stated or referenced in Table B.3 of this recommended practice shall be evaluated considering the criteria provided in 5.3.4.2 of IEEE/EIA 12207.0-1996.

**Table B.3— Coverage of specific SRD requirements by IEEE Std 830-1998**

IEEE/EIA 12207.1-1997 specific content	Corresponding clauses of IEEE Std 830-1998	Additions to requirements of IEEE Std 830-1998
a) Generic description information	See Table B.2	—
b) System identification and overview	5.1.1 Scope	—
c) Functionality of the software item including: – Performance requirements – Physical characteristics – Environmental conditions	5.3.2 Functions 5.3.3 Performance requirements	Physical characteristics and environmental conditions should be provided.
d) Requirements for interfaces external to software item	5.3.1 External interfaces	—
e) Qualification requirements	—	The requirements to be used for qualification testing should be provided (or referenced).
f) Safety specifications	5.2.4 Constraints	—
g) Security and privacy specifications	5.3.6.3 Security	—
h) Human-factors engineering requirements	5.2.3 User characteristics 5.2.1.2 User interfaces	—
i) Data definition and database requirements	5.3.4 Logical data base requirements	—
j) Installation and acceptance requirements at operation site	5.2.1.8 Site adaptation requirements	Installation and acceptance requirements at operation site
k) Installation and acceptance requirements at maintenance site	—	Installation and acceptance requirements at maintenance site
l) User documentation requirements	—	User documentation requirements
m) User operation and execution requirements	5.2.1.7 Operations	User execution requirements

**Table B.3—Coverage of specific SRD requirements by IEEE Std 830-1998 (continued)**

IEEE/EIA 12207.1-1997 specific content	Corresponding clauses of IEEE Std 830-1998	Additions to requirements of IEEE Std 830-1998
n) User maintenance requirements	5.3.6.4 Maintainability	—
o) Software quality characteristics	5.3.6 Software system attributes	—
p) Design and implementation constraints	5.2.4 Constraints	—
q) Computer resource requirements	5.3.3 Performance requirements	Computer resource requirements
r) Packaging requirements	—	Packaging requirements
s) Precedence and criticality of requirements	5.2.6 Apportioning of requirements	—
t) Requirements traceability	4.3.8 Traceable	—
u) Rationale	5.2.5 Assumptions and dependencies	—
Items a) through f) below are from 6.22.4	—	Support the life cycle data objectives of Annex H of IEEE/EIA 12207.0-1996
a) Support the life cycle data objectives of Annex H of IEEE/EIA 12207.0-1996		
b) Describe any function using well-defined notation	4.3 Characteristics of a good SRS	—
c) Define no requirements that are in conflict	4.3 Characteristics of a good SRS	—
d) User standard terminology and definitions	5.1.3 Definition	—
e) Define each unique requirement one to prevent inconsistency	4.3 Characteristics of a good SRS	—
f) Uniquely identify each requirement	4.3 Characteristics of a good SRS	—

### B.3.4 Compliance with life cycle data objectives

In addition to the content requirements, life cycle data shall be managed in accordance with the objectives provided in Annex H of IEEE/EIA 12207.0-1996.

## B.4 Conclusion

The analysis suggests that any SRS complying with this recommended practice and the additions shown in Table B.2 and Table B.3 also complies with the requirements of an SRD in IEEE/EIA 12207.1-1997. In addition, to comply with IEEE/EIA 12207.1-1997, an SRS shall support the life cycle data objectives of Annex H of IEEE/EIA 12207.0-1996.

## **ANEXO B**

### **Documento de visión**

---

**Visión**

**Versión de la plantilla <1.0>**

**Revisión histórica**

Fecha	Versión	Descripción	Autor
<dd/mm/aa>	<x.x>	<detalles>	<nombre>

**Índice**

[1. Introducción](#)

[1.1 Referencias](#)

[2. Posicionamiento](#)

[2.1 Definición de la necesidad](#)

[3. Descripción de los stakeholders y usuarios](#)

[3.1 listado de stakeholders](#)

[3.2 listado de usuarios](#)

[3.3 Entorno de usuario](#)

[3.4 listado de las necesidades claves de los stakeholders y usuarios](#)

[3.5 Alternativas y competencia](#)

[4. Generalidades del producto](#)

[4.1 Perspectiva del producto](#)

[4.2 Restricciones y Supuestos](#)

[5. Características del producto](#)

[6. Otros requisitos del producto](#)

---

## Visión

### 1. Introducción

*[El propósito de este documento es recoger, analizar y definir necesidades y características de alto nivel del sistema. Este documento se enfoca en las capacidades necesitadas por los participantes y los usuarios finales, y en el porqué estas necesidades existen. Los detalles de cómo el sistema satisface estas necesidades son detallados en las especificaciones de casos de uso y suplementarias.]*

*[La introducción del documento de Visión brinda una descripción general de todo el documento y se compone por los siguientes apartados:.]*

#### **Propósito del documento**

*[Especifica el propósito de este documento de Visión.]*

#### **Alcance del documento**

*[Una breve descripción del alcance del documento de Visión, los proyectos con los que está asociado y cualquier otra cosa que sea afectada o influenciada por este documento.]*

#### **Definiciones, Acrónimos, y Abreviaturas contenidas en el documento**

*[Esta subsección incluye las definiciones de todos los términos, acrónimos, y abreviaturas requeridas para interpretar apropiadamente el documento de Visión. ]*

#### **Referencias a otros documentos**

*[Esta subsección incluye una lista completa de todos los documentos referenciados en el documento de Visión. ]*

### 2. Posicionamiento

#### 2.1 Definición de la Necesidad

*[Esta subsección se menciona "la" necesidad general, que pretende afrontar el proyecto ]*

<b>Para la necesidad</b>	
<b>Que afecta a:</b>	
<b>Y cuyo impacto es:</b>	
<b>Una solución adecuada debería proveer:</b>	

### 3. Descripción de Stakeholders y usuarios

*[Para satisfacer efectivamente las necesidades de los participantes y usuarios, es necesario identificar e involucrar a todos los participantes como parte del proceso de Modelado de Requerimientos. También es necesario identificar los usuarios del sistema para tener la seguridad de que la comunidad de participantes los representa adecuadamente. Esta sección brinda un perfil de los participantes y los usuarios involucrados en el proyecto, como así también de los problemas clave que ellos consideran incluir en la solución propuesta. Este documento no describe los pedidos o requerimientos de manera específica. En su lugar, describe el contexto y la justificación de porqué los requerimientos son necesarios.]*

#### 3.1 Listado de Stakeholders

*[Presenta una lista de todos los participantes identificados.]*

<b>Nombre</b>	<b>Descripción</b>	<b>Responsabilidades</b>

### 3.2 Listado de usuarios

*[Presenta una lista de todos los usuarios identificados.]*

Nombre	Descripción	Responsabilidades	Stakeholders

### 3.3 Entorno de usuario

*[Detalle el ambiente de trabajo de los usuarios finales.]*

### 3.4 Necesidades claves de los stakeholders o usuarios

*[Liste los problemas más importantes con soluciones existentes tal cómo lo perciben los participantes o usuarios. Clarifique los siguientes aspectos:*

- ¿Cuáles son las razones para este problema?*
- ¿Cómo se soluciona actualmente?*
- ¿Qué soluciones quiere el participante?*

*Es importante comprender la importancia relativa que el participante o usuario le da a la resolución de cada problema.]*

Necesidad	Prioridad	Impacto ocasionado	Solución actual	Solución propuesta

### 3.5 Alternativas y competencia

*[Identifique las alternativas detectadas por los participantes. Estas pueden incluir la compra de un sistema ya desarrollado, construir una solución propia o simplemente mantenerle status quo.]*

## 4. Generalidades del Producto

*[Esta sección brinda una vista de alto nivel de las capacidades del producto, de las interfaces a otras aplicaciones y de las configuraciones del sistema.]*

### 4.1 Perspectiva del producto

*[Esta subsección del documento de Visión presenta el sistema en perspectiva frente a otros sistemas relacionados y al entorno del usuario. Si el sistema es independiente y totalmente autocontenido, méncionelo aquí. Si el sistema es una componente de un sistema mayor, en esta subsección se describirá cómo interactúan estos sistemas y se identificará las interfaces relevantes entre ellos. Una manera sencilla de mostrar las componentes más importantes del sistema principal, las interconexiones y las interfaces externas es con un diagrama de bloques.]*

### 4.2 Restricciones y supuestos

*[Liste cada uno de los factores que afectan las características establecidas en el documento de Visión. Liste las suposiciones que si fueran cambiadas alterarían este documento.] [Destaque cualquier restricción de diseño, restricción externa u otra dependencias.]*

## 5. Características del producto

*[Liste y describa brevemente las características del sistema. Las características son las capacidades de alto nivel del sistema que son necesarias para entregar los beneficios a los usuarios. Cada característica es un servicio deseado externamente que típicamente requiere una serie de entradas para lograr el resultado esperado.]*

## 6. Otros requisitos del producto

*[Utilice esta sección para detallar los requerimientos no funcionales.]*

## **ANEXO C**

### **Necesidades del cliente**

Sesión # \_\_\_\_\_

Entrevistador: \_\_\_\_\_

Representante del cliente: \_\_\_\_\_

Modulo: \_\_\_\_\_

Sub-modulo: \_\_\_\_\_

Listado de necesidades:

Necesidad	Prioridad	Cómo se resuelve	Notas	Requerimiento	Fecha de entrega

Acuerdos: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

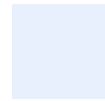
Fecha de siguiente sesión: \_\_\_\_\_





## **ANEXO D**

### **Documentación de requerimientos**



REQ- <i>[Identificador único]</i>	<i>[Nombre descriptivo del requerimiento]</i>	
FECHA	<i>[Fecha de especificación]</i>	
VERSION	<i>[Versión del documento de requerimientos]</i>	
AUTORES	<i>[Lista de claves, que se definieron en la plantilla de roles y responsabilidades]</i>	
REQUERIMIENTOS ASOCIADOS	<i>[Lista de claves de los requerimientos que se encuentran relacionados con este requerimiento]</i>	
DESCRIPCION	<i>[Describir detalladamente el requerimiento siguiendo este patrón:                  &lt;&lt;objeto o función&gt;&gt; + deberá + &lt;&lt;acción a realizar&gt;&gt; + &lt;&lt;condiciones&gt;&gt;                  O                  Se deberá poder + &lt;&lt;acción a realizar&gt;&gt; + &lt;&lt;objeto o función&gt;&gt; + &lt;&lt;condiciones&gt;&gt;</i>	
SECUENCIA DE PASOS	PASO	ACCION
	P1	<i>[Descripción del paso que se realiza, indicando quién lo ejecuta, colocando la clave que se le asigno en la plantilla de roles y responsabilidades]</i>
	P2	<i>[Describir las diferentes acciones y condiciones, en este y los siguientes pasos]</i>
	P3	
	...	<i>...[Todos los pasos necesarios]</i>
EXCEPCIONES	EXCEPCION	ACCION
	E1	<i>[Describir las posibles excepciones que se pueden presentar, mientras se ejecutan los pasos descritos antes]</i>
	E1	
	...	<i>...[Todas las posibles excepciones]</i>
PRIORIDAD	<i>[Nivel de importancia]</i>	
COMENTARIOS	<i>[Comentarios adicionales]</i>	
FIRMA DE CONFORMIDAD	<i>[Firma de autorización del cliente]</i>	

## **ANEXO E**

### **Documentación al finalizar**



Autor	Fecha de creación	Versión
Quien crea el documento	dd/mm/yyyy	1.0
Modifica	Fecha de modificación	
Quien modifica el documento	dd/mm/yyyy	

**Modulo:** Nombre del módulo, por ejemplo: Control Becas.

**Proceso:** Nombre del proceso, por ejemplo: Cambio de periodo.

**Descripción:** Descripción breve del módulo o proceso.

**Para acceder:** Escribir la ruta de acceso al módulo o proceso utilizando los caracteres “->” y utilizando las etiquetas que se muestran en el sistema, por ejemplo: Pantalla principal->Control Becas->Cambio de periodo.

**Funcionamiento:**

Enumerar los pasos para el funcionamiento del módulo o proceso.

**Condiciones:**

Enumerar las condiciones para el buen funcionamiento del módulo o proceso.