PASCAR: PROTOTIPO DE APLICACIÓN SOFTWARE PARA LA CAPTURA, ESPECIFICACIÓN Y DOCUMENTACIÓN DE REQUERIMIENTOS DE SOFTWARE, UTILIZANDO EL ESTÁNDAR IEEE 830.

ADRIANA ROCÍO PÉREZ ROJAS

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÌSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2005

PASCAR: PROTOTIPO DE APLICACIÓN SOFTWARE PARA LA CAPTURA, ESPECIFICACIÓN Y DOCUMENTACIÓN DE REQUERIMIENTOS DE SOFTWARE, UTILIZANDO EL ESTÁNDAR IEEE 830.

ADRIANA ROCÍO PÉREZ ROJAS

Trabajo de grado para optar el título de Ingeniera de Sistemas

Director
Ing. JOSÉ CÁRCAMO SEPÚLVEDA
Magíster en Informática

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÌSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2005

Con todo mi cariño
a mis padres Ulises y Maria Elsa,
a mis hermanas.

A.R.P.R

AGRADECIMIENTOS

La autora expresa sus agradecimientos:

A Dios por darme la fortaleza y abrirme el camino para llegar hasta aquí

A mi familia por su confianza y ayuda, en especial a mi madre por su compañía, comprensión y apoyo.

Al profesor José Cárcamo Sepúlveda por su gran disposición, guía y pronta colaboración durante el desarrollo del proyecto.

Al Ingeniero Edward José Beltrán Lozano por su atención, orientación y paciencia al brindarme asesoría en el proyecto.

Y a todos aquellos que intervinieron en la culminación de este trabajo de grado… muchas gracias…

RESUMEN

TITULO
PASCAR: PROTOTIPO DE APLICACIÓN SOFTWARE PARA LA CAPTURA, ESPECIFICACIÓN Y DOCUMENTACIÓN DE REQUERIMIENTOS DE SOFTWARE, UTILIZANDO EL ESTÁNDAR IEEE 830[*].

AUTOR
Adriana Rocío Pérez Rojas[**]

PALABRAS CLAVE
Ingeniería de Requerimientos, Estándar IEEE 830, Requisitos, Especificación de Requisitos, Ingeniería del Software.

DESCRIPCION
La Ingeniería del Software se ha encargado de brindar metodologías y estándares para mejorar el proceso de desarrollo del software. Existen productos software en el mercado que han adaptado algunas de éstas metodologías para intervenir en el proceso del ciclo de vida del software y apoyar la industria del software; pero aún existen muchos problemas relacionados con el desarrollo de soluciones software porque muchas de las empresas que pertenecen a ésta industria, desconocen esos procesos o no están en la capacidad de adquirir las soluciones del mercado, desarrollando sus productos de manera artesanal o inadecuada.

PASCAR es un prototipo software que utiliza como guía el estándar IEEE 830 y ha sido planteado como herramienta alternativa para ser utilizada en las actividades de la Ingeniería de Requerimientos, ayudando a la gestión de requisitos dentro de un proyecto software. Este proyecto consta de dos aplicaciones: un prototipo de escritorio que facilita la captura y administración de requerimientos, para la generación del documento de Especificación de Requisitos de Software (SRS) y un prototipo móvil, que puede utilizarse como herramienta auxiliar para capturar y administrar los requerimientos de software en un dispositivo móvil (Pocket PC) desde el lugar que lo necesite; el primero es independiente del segundo, pero se ha adicionado éste segundo prototipo al proyecto como valor agregado, para darle al prototipo completo una característica de movilidad.

---

[*] Proyecto de Grado – Modalidad Investigación

[**] Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería de Sistemas , Director: José Cárcamo Sepúlveda

ABSTRACT

TITLE
PASCAR: PROTOTYPE OF SOFTWARE APPLICATION FOR SOFTWARE REQUIREMENTS CAPTURE, SPECIFICATION AND DOCUMENTATION, USING IEEE 830 STANDARD*.

AUTHOR
Adriana Rocío Pérez Rojas**

KEYWORDS
Requirements Engineering, IEEE 830 Standard, Requirements, Requirements Specification, Software Engineering.

DESCRIPTION
Software Engineering is in charge of offer methodologies and standards to improve the software development process. There are software products in the market that have adapted some methodologies to take part in the software life cycle process and support the software industry; but yet there are many problems related with the software solutions development because many enterprises in this software industry don't know those processes, or they don't have purchasing power to purchase market solutions, making their products in way inadequate.

PASCAR is a software prototype that uses as guide the IEEE 830 standard and it has been proposed like alternative tool to be used in the requirements engineering activities, supporting requirements management inside software project. This project consist of two applications: a desktop prototype which support the requirements capture and management, in order to generate the software requirements specification (SRS) document; and a mobile prototype which is used like auxiliary tool to capture and management the software requirements in a mobile device (Pocket PC) from place where it be necessary; the first one is independent of the second one, but the second prototype is added to project like aggregate value to give a mobility characteristic to complete prototype.

---

* Degree Project

** Faculty of Physical-Mechanical Engineering, Systems Engineering School, Director: José Cárcamo Sepúlveda

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

INTRODUCCIÓN

Actualmente la industria del software muestra un crecimiento casi desmedido, sin embargo, no siempre satisface la demanda existente en el mercado y usualmente se encuentra una alta incidencia de fallos en los proyectos de software que son contratados para brindar “soluciones”. La Ingeniería del Software se ha encargado del estudio y la difusión de metodologías y estándares que optimicen el proceso del desarrollo de software y contribuyan a la elaboración de software de calidad. Una de las áreas de la Ingeniería de Software involucrada en el mejoramiento de los procesos del ciclo de vida del software es la Ingeniería de Requerimientos, que interviene en la primera fase de desarrollo del software, la fase de requerimientos, que es considerada una de las más importantes y de las cuales depende el éxito o fracaso de un proyecto software.

En éste proyecto se describen algunos de los problemas que sufre la elaboración de proyectos software, que parten de una mala especificación de requerimientos y se propone un prototipo software que toma como base un estándar que promueve la Ingeniería del Software: el estándar IEEE 830 para la especificación de requerimientos de software, como una manera de propender por la difusión de procesos de desarrollo que mejoren la elaboración de productos en la industria del software.

El informe de éste proyecto se encuentra organizado por capítulos de la siguiente manera: en el primer capítulo se encuentra la definición del problema, los objetivos propuestos, el alcance e impacto del proyecto; en el segundo capítulo se encuentra el marco teórico del proyecto que incluye un apartado sobre Ingeniería del Software, otro de Ingeniería de Requisitos y para finalizar la descripción del estándar IEEE 830 utilizado en el proyecto; el tercer capítulo es un apartado muy breve que contiene los aspectos técnicos involucrados en el proyecto; en el cuarto capítulo se encuentra el diseño y la metodología aplicada al proyecto y la funcionalidad del prototipo; en el quinto capítulo se encuentran las conclusiones seguidas por las recomendaciones, referencias bibliográficas y anexos.

1. PLANTEAMIENTO DEL PROBLEMA

En éste capítulo se encuentra la definición del problema del cual fue objeto éste proyecto, describiendo algunos de sus síntomas y efectos, los objetivos trazados, los alcances propuestos y el impacto estimado para el desarrollo de éste proyecto.

1.1. DEFINICIÓN DEL PROBLEMA

En el área de la Ingeniería del Software se define el "Ciclo de Vida" del software como el conjunto de procesos y técnicas aplicadas al desarrollo de un producto software, desde que se concibe la necesidad de crearlo, hasta que termina su explotación. Dentro del ciclo de vida se encuentran varios procesos básicos, entre los cuales está la "Definición de Requisitos", que comprende aquellas actividades que traducen las necesidades del cliente del producto, en un modelo consistente, que define la estructura y el comportamiento del software a desarrollar.

En los últimos años, las organizaciones han aumentado la demanda del software, en su afán por actualizarse y administrar de manera eficiente y competitiva su información; pero muchas de ellas se han encontrado con productos software que no cumplen con sus especificaciones, son incompletos y de mala calidad. Esta situación puede ser consecuencia de un problema que se origina en las fases iniciales del ciclo de vida del software: la especificación deficiente de requerimientos dentro de un proyecto software. Para este problema, podemos encontrar los siguientes síntomas y efectos:

- No hay una comprensión del dominio de la aplicación por parte del equipo que esta involucrado en el proyecto, y por tanto no se define claramente el resultado que se desea obtener del producto software.

- Falta establecer una separación entre los diferentes niveles de descripción de los requerimientos (como requerimientos de diseño, de interfaces, de seguridad, restricciones, entre otros), y se termina por agrupar diferentes requerimientos en uno solo u omitiéndose requerimientos que pueden ser importantes.

- La especificación de requerimientos se hace de forma incompleta, imprecisa, desordenada e inconsistente, entre otras cosas, por falta de preparación del personal que esta encargado de esta tarea, porque no se conocen ni utilizan estándares o formatos para la captura, especificación, organización y documentación de requerimientos, que facilite la administración de los

mismos, en especial, cuando se generan cambios en las especificaciones a lo largo del proyecto. Esto puede hacer que el proyecto software sufra retrasos en el cronograma e incremente los costos, pues no hay una apropiada documentación de los requerimientos que facilite su gestión.

- Las pruebas del software fallan. La falta de documentación de requisitos, puede ocasionar que no se planeen buenas pruebas para el software antes que el producto sea entregado, comprometiendo su calidad.

- Cuando no se provee la documentación detallada de requerimientos de un producto software, se dificulta su mantenimiento, y la obtención de nuevas versiones y actualizaciones del mismo.

Estos son los principales síntomas y efectos que ocasionan una deficiencia notable en la especificación y administración organizada de requerimientos dentro de un proyecto software. Debido a las fallas e inconsistencias en el producto software, se plantea la necesidad que las organizaciones que ofrecen sus servicios como desarrolladoras de éste tipo de proyectos, adopten esquemas más formales para la especificación de requerimientos de software, en pro de mejorar la documentación y administración de requisitos y por tanto, ofrecer un producto que cumpla con las expectativas del cliente.

Para mejorar la fase de especificación de requisitos se han desarrollado estándares y paquetes software, que pretenden aportar un mayor control a esta fase del ciclo de vida del software, a través de una administración organizada de requerimientos, que van desde su definición, clasificación y seguimiento, hasta su actualización y mantenimiento. Algunas empresas desarrolladoras de software desconocen esto y llevan poco o ningún control sobre sus productos. Otras crean su propios formatos de especificación o si pueden, adquieren herramientas CASE disponibles en el mercado. Dentro de las soluciones propuestas en el mercado podemos encontrar algunas como: IRqA – Integral Requisite Analyzer, desarrollado por TCP Sistemas e Ingeniería, que es una herramienta CASE de Ingeniería de Requisitos especialmente diseñada para soportar las actividades realizadas en el proceso de especificación de sistemas, ayudando a los analistas a capturar, gestionar, analizar los requisitos y a trazar los mismos a lo largo de las distintas fases del desarrollo; DOORS, de Telelogic que utiliza un tratamiento integrado de la trazabilidad de los requisitos, es decir, se basa en las relaciones existentes entre requisitos por motivos de jerarquía o particionado y permite hacer análisis de impacto top-down (de especificación a implementación) y down-top (de implementación a especificación); RUP – Racional Unified Process de Rational Software Corporation, es una metodología para el desarrollo de proyectos software, que incluye un buen soporte para la administración de requerimientos y para las demás fases del ciclo de vida del software; entre

otras encontramos RTM, XTie-RT y RequisitePro, como soluciones que aportan mejoras a la fase de análisis y especificación de requerimientos de software.

En este sentido surge la idea de elaborar un prototipo software, enfocado en la organización y administración de los requisitos de software, que sirva de herramienta al analista de sistemas para la captura y documentación de requerimientos y que preste soporte a las siguientes fases del ciclo de vida del software, en la elaboración de productos software con mayor aceptación.

1.2. OBJETIVOS

1.2.1. Objetivo General

Diseñar e implementar un prototipo software para la captura, especificación y documentación de requerimientos de software, utilizando el Estándar IEEE 830.

1.2.2. Objetivos Específicos

- Diseñar e implementar un prototipo software para la captura, especificación, documentación y administración de requerimientos de software, desde un computador de escritorio, que será organizado en los siguientes módulos principales:

  - Módulo de Administración de Proyectos
  - Módulo de Especificación de Proyectos
  - Módulo de Especificación de Requerimientos
  - Módulo de Documentación
  - Módulo de Mantenimiento

- Diseñar e implementar un prototipo software para la captura y seguimiento de requerimientos de software, desde un Asistente personal digital (PDA).

- Diseñar e implementar un software de sincronización (conduit[1]) entre la aplicación de escritorio y la aplicación móvil del PDA, para la transferencia de datos entre los dos dispositivos (Computador de escritorio y PDA).

[1] Es un proceso de comunicación entre la aplicación de escritorio y la aplicación móvil del PDA, que implica una fase de sincronización en la cual se intercambian y actualizan datos entre el dispositivo de escritorio y el dispositivo móvil (PDA).

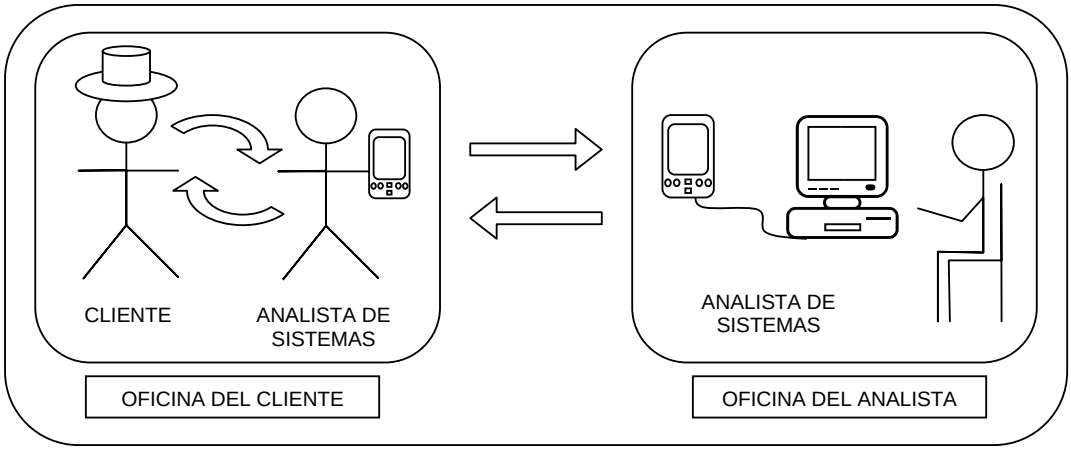


Figura 1. Representación del prototipo PASCAR

1.3. ALCANCES

El proyecto PASCAR pretende ser un prototipo software que ayude a los ingenieros de sistemas u otros agentes involucrados, en la administración de requerimientos de software, no siendo así utilizado como herramienta única e infalible, pero sí que pueda utilizarse a la par con otras herramientas que den soporte a las tareas de diseño, de un modo más formal y organizado.

1.4. IMPACTO

- En el ámbito científico se pretende:

  - Fomentar el formalismo en la especificación de requerimientos de software, utilizando un estándar de aceptación internacional (como es el Std. IEEE 830), a través del desarrollo de una herramienta que sirva como guía para la documentación de requisitos.

  - Incentivar la línea de investigación en Ingeniería del Software, en busca de mejorar los procesos utilizados en las fases de desarrollo del software, y en este caso, la especificación de requisitos (así mismo, la mejora de otros procesos que propendan por la obtención de productos software de mayor calidad).

- En el ámbito social se pretende:

  - Orientar las actividades realizadas por los analistas de sistemas en el proceso de captura, especificación, gestión y documentación de requerimientos de software, con el uso de una aplicación software, que le permita administrar los diferentes proyectos de desarrollo de software que tenga a su cargo, de una manera más organizada.

  - Proveer a los clientes de productos software medios de verificación de sus especificaciones a través de la documentación de requerimientos de software, de modo que puedan corregir a tiempo las inconsistencias que se presenten. Así, el producto software se acercará más a lo que el cliente necesita, proporcionándole una mayor satisfacción a éste.

- En el ámbito económico se pretende:

  - Reducir los costos que se acrecientan cuando se presentan nuevos requerimientos a medio camino o modificación de los que ya existen, pues debido a la documentación deficiente de requisitos que se maneja en algunos proyectos software, estos cambios se vuelven tediosos y surgen retrasos en el cronograma (y consecuentemente se incrementan los costos).

2. MARCO TEÓRICO

En el marco de éste capítulo encontramos una introducción a la ingeniería del software, seguida por una descripción del proceso de la ingeniería de requisitos, algunas técnicas utilizadas en la ingeniería de requisitos y termina con la descripción del estándar IEEE 830 utilizado en el proyecto como guía para la especificación de requisitos de software.

2.1. INGENIERÍA DEL SOFTWARE

La ingeniería del software implica "el estudio y la aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software". Dentro de las muchas definiciones que podremos encontrar acerca de la ingeniería del software, R. Pressman[2] cita ésta que está dada por el IEEE (Estándar IEEE 610.12 de Ingeniería del Software). Se percibe la obtención de un objetivo específico y claro: desarrollar soluciones software eficientes.

Pressman nos plantea la ingeniería del software como una tecnología multicapa, que comprende un proceso, métodos técnicos y de gestión, y herramientas, todas soportadas sobre un enfoque de calidad, que facilita el control del desarrollo del software de manera productiva; Los métodos nos ayudan a clarificar el "cómo" construir técnicamente el software, e implica tareas de planificación, análisis de requisitos, diseño, codificación, pruebas y mantenimiento; Las herramientas proporcionan un soporte automático o semi-automático para los métodos y el proceso es la unión de los métodos y las herramientas que facilita un desarrollo de software racional y oportuno. A modo de generalización, se puede decir que el trabajo de la ingeniería del software se divide en tres fases que involucran métodos, herramientas y procedimientos, y se describen a continuación:

Fase	Enfoque	Descripción
Definición	Qué	Identificación de la información que ha de ser procesada para definir correctamente el sistema. Definición de requisitos del sistema y del software.

[2] PRESSMAN, Roger. Ingeniería del Software, un enfoque práctico, Quinta Edición. McGraw Hill, España, 2002. Capítulo 2, Pág. 14.

Desarrollo	Cómo	Definición del diseño del software, desarrollo e implementación y pruebas del software.
Mantenimiento	Cambio	Detección de errores, adaptaciones a los cambios del entorno y de los requisitos del cliente.

Tabla 1. Fases del Trabajo de la Ingeniería del Software

Esto involucra por sí mismo un despliegue de metodologías para el desarrollo del software que se encuentran definidas para soportar las actividades propias del ciclo de vida del software. Algunas de ellas son:

1. **Modelo Lineal Secuencial (Cascada Pura).** Es la base para otros modelos más efectivos de ciclo de vida. Un proyecto software basado en esta metodología progresa a través de una secuencia ordenada de pasos, que no permite el avance a la siguiente actividad hasta tanto no se haya revisado y terminado la actual. El modelo define las siguientes actividades para el ciclo de vida: análisis de requerimientos, diseño, codificación y pruebas. Éste modelo ha sido transformado en la metodología de cascadas modificadas a través del solapamiento de las fases descritas, para permitir un mayor avance a través del proyecto. El modelo lineal secuencial, también se ha utilizado en el modelo **DRA** (Desarrollo Rápido de Aplicaciones), utilizando una construcción basada en componentes y permitiendo desarrollos en periodos cortos de tiempo.

2. **Modelo de Construcción de Prototipos.** Este modelo se centra en el desarrollo de los aspectos más visibles del software a través de un diseño rápido, que da entrada a la definición de requerimientos más específicos a medida que avanza el proyecto. Se basa en el desarrollo de prototipos de software que utilizan la realimentación proporcionada por el cliente, hasta el desarrollo del producto final. Las tareas propuestas para este modelo son: análisis de requisitos (dentro de un concepto inicial del software), diseño e implementación del prototipo inicial, refinar el prototipo hasta que sea aceptable (realimentación) y completar y entregar el prototipo como producto final al cliente.

3. **Modelos Evolutivos.** Son modelos iterativos que permiten desarrollar versiones de software cada vez más completas y funcionales. Dentro de éstas metodologías podemos encontrar el Modelo Incremental que combina el modelo en cascada pura y el de construcción de prototipos a través de la entrega de incrementos durante el tiempo de desarrollo del proyecto. Otro modelo que se encuentra dentro de ésta categoría es el

Modelo en Espiral que combina la construcción de prototipos con el modelo orientado a riesgos que proporciona el modelo lineal secuencial.

Se puede encontrar otra forma de clasificación de metodologías y modelos para el proceso de desarrollo del software, que hacen referencia a los mencionados anteriormente o son híbridos de unos y otros, pero que en esencia buscan dar soporte a las actividades del ciclo de vida de desarrollo de software.

2.2. INGENIERÍA DE REQUISITOS (IR[3])

"La parte más difícil en la construcción de sistemas software es decidir precisamente qué construir. Ninguna otra parte del trabajo conceptual es tan dificultosa como establecer los requerimientos técnicos detallados, incluyendo todas las interfaces con humanos, máquinas y otros sistemas software. Ninguna otra parte del trabajo puede perjudicar tanto el resultado final si es realizada en forma errónea. Ninguna otra parte es tan dificultosa de rectificar posteriormente."[4]

Conforme a esto, se reitera que la parte más importante para la definición y desarrollo de un proyecto software es la especificación de requerimientos de manera que se pueda realizar la trazabilidad de los mismos hasta las pruebas finales y puesta en marcha del desarrollo realizado. En cuanto a éste proyecto se refiere, encontramos dentro de la ingeniería del software, una disciplina que se ha desarrollado para cubrir las fases iniciales del ciclo de vida del software: la ingeniería de requisitos, que involucra todas las actividades, técnicas y metodologías aplicadas a la definición y especificación de las necesidades de los clientes de un proyecto software

La ingeniería de requisitos se ha encargado de crear mecanismos que ayuden a la especificación, validación y gestión de requisitos de software para que éstos puedan convertirse en sistemas que operen como el cliente precisa. Se necesita entonces una buena especificación de requisitos, ya que ésta es la que determina que hará el sistema a desarrollar, qué restricciones se deben tener en cuenta en el diseño y de ello depende la satisfacción del usuario final. Roger Pressman hace referencia al proceso de la ingeniería de requisitos, que puede describirse, según Ian Sommerville[5], en los siguientes pasos:

---

[3] IR- Ingeniería de Requisitos / Ingeniería de Requerimientos

[4] Frederick P. Brooks, Jr., The Mythical Man-Month, Addison-Wesley, 1995.

[5] Citado en PRESSMAN, Roger. Ingeniería del Software, un enfoque práctico, Quinta Edición. McGraw Hill, España, 2002. Capítulo 10, Pág.. 171-175 y Capítulo 11, Pág. 181.

2.2.1. Identificación de Requisitos

El propósito de ésta actividad es comprender las verdaderas necesidades del negocio a través de una buena comunicación entre el equipo desarrollador del proyecto software y el cliente. Luego, los objetivos trazados van desde la comprensión del área del negocio del cliente, la evaluación de las necesidades planteadas hasta la propuesta de una solución favorable y de calidad para resolver el problema.

Es ésta fase del análisis del problema se deben tener en cuenta ciertos aspectos que ayuden a garantizar un acuerdo entre las partes involucradas en el proyecto:

- Identificar quiénes son realmente los involucrados y afectados con el desarrollo del producto software de manera directa e indirecta, es decir, determinar quién es el usuario que usara el sistema desarrollado, quién se encargará de su desarrollo, soporte y mantenimiento, quién se verá beneficiado con la elaboración del proyecto software.
- Entender el problema y vislumbrar desde los diferentes puntos de vista de los involucrados, la solución más factible tomando en cuenta tanto la perspectiva de cliente como la perspectiva técnica del equipo de desarrollo.
- Elaborar un glosario de términos comunes que facilite la comunicación entre las partes y sirva como herramienta para evitar definiciones ambiguas y que todos los interesados se entiendan con el mimo lenguaje.
- Definir los límites y las restricciones (de tiempo, presupuesto, del entorno, entre otras) que se deben tener en cuenta antes de la planificación del desarrollo del proyecto para tener un norte claro del alcance del proyecto.

En síntesis, en este paso los ingenieros de sistemas deben organizar una serie de reuniones o encuentros (pueden utilizarse entrevistas, cuestionarios, encuestas u otras metodologías para recolección de información) con los clientes/usuarios involucrados con el producto software a desarrollar, para definir qué es lo que se necesita obtener del producto y superando los problemas que suelen presentarse por la mala definición del alcance del proyecto, por problemas de comprensión y volatilidad de los requerimientos a través del tiempo. El resultado de estos encuentros debe ser una definición clara de las necesidades del cliente, los objetivos y el alcance del sistema.

2.2.2. Análisis y Negociación de Requisitos

Es necesario realizar una depuración de los requerimientos identificados y evaluarlos de manera objetiva, de modo que se obtengan resultados factibles, coherentes y completos, pero aterrizados al cliente dentro de los limites establecidos. En esta parte se deben tener en cuenta los siguientes aspectos:

- Identificar los requerimientos que se definieron de forma ambigua, inconsistente e incompleta, de modo que éstos no ocasionen problemas más adelante.
- En la medida que se analice la importancia de cada requerimiento es bueno clasificarlo e identificarlo de acuerdo a la prioridad que éste tenga dentro del proyecto y el modelo del negocio del cliente.
- Evaluar las factibilidades de la implementación de los requerimientos definidos, de modo que puedan satisfacerse con la tecnología existente, que se acoplen con la operación del negocio, con el presupuesto disponible, entre otros.

En decir, se debe realizar una organización y agrupación de los requisitos para obtener un análisis de cada requisito en relación con los demás, con las necesidades del cliente, con los objetivos propuestos y con la consistencia, claridad y completitud necesarias. Se debe hacer una negociación iterativa con el cliente para poder realizar un filtrado de los requisitos pertinentes para el producto que se espera obtener, de modo que se logre un beneficio mutuo.

2.2.3. Especificación de Requisitos (SRS[6])

En ésta fase se obtiene el resultado final de las actividades de análisis y evaluación de requerimientos. Durante la especificación de requerimientos de software (SRS) se obtiene un documento que contiene plasmado la descripción completa del sistema que se va a desarrollar a partir de la definición del alcance del proyecto, las funcionalidades del sistema, los requerimientos de hardware, software, interfaces, entre otros.

Ésta especificación se utiliza en las fases de diseño, construcción y pruebas del software y es utilizado como parte del contrato y medio de comunicación entre todas las partes involucradas durante el desarrollo del proyecto software. Para el cliente, esta documentación describe sus necesidades, para el equipo de desarrollo indica qué es lo que debe elaborarse, para el gerente de proyecto es una medida de control y avance del sistema, y para el equipo de pruebas la base de la calidad del sistema desarrollado.

Resumiendo, es en éste paso donde se obtiene el producto final del análisis de requisitos que realiza el ingeniero de sistemas (o ing. de requisitos). Puede obtenerse como resultado un documento escrito, combinado con modelos gráficos y/o matemáticos que describa las funciones y características del producto software a desarrollar, manteniendo las características de

[6] SRS- Especificación de Requerimientos de software

consistencia, verificabilidad, factibilidad, no ambigüedad, trazabilidad y precisión.

2.2.4. Modelado del Sistema

En este paso se evalúan los componentes del sistema y sus relaciones, cómo se reflejan los requisitos y cómo se ha concebido "estéticamente" el sistema, teniendo en cuenta que deben ser representados desde la perspectiva del cliente y cómo éste percibe sus necesidades.

2.2.5. Validación y Gestión de Requisitos

La validación realizada en esta parte del proceso, tiene como fin verificar que los requerimientos definidos en la SRS son los que en realidad demanda el cliente. Por ejemplo se debe cuestionar si todas las funciones requeridas por el cliente están incluidas en la SRS, si no existen problemas de consistencia, ambigüedad, claridad, factibilidad, verificabilidad y trazabilidad.

La gestión de requisitos es una actividad que se mantiene durante todo el desarrollo del proyecto y está sujeta a la volatilidad de los requerimientos. Un requerimiento se transforma con el tiempo debido a cambios que ocurren en el planteamiento del problema, en el ambiente, en las percepciones del cliente o debido a una definición deficiente del mismo. Por esto es importante llevar un control de las versiones de los requerimientos y tener claras las dependencias y relaciones que tienen con otros requerimientos de modo que no se pierda la integridad necesaria para las etapas posteriores en el desarrollo del proyecto.

Podemos concluir entonces, que la calidad del producto software se asegura cuando se valida la especificación de requisitos pactada y se revisa que los requisitos del sistema han sido establecidos sin ambigüedad ni inconsistencia, y que los errores hayan sido corregidos. Con la gestión de requisitos se asegura cierto control de los requisitos y los cambios que se presenten en un momento dado.

2.3. TÉCNICAS USADAS EN LA INGENIERÍA DE REQUISITOS

Dependiendo del proyecto software a desarrollar, pueden utilizarse técnicas de la IR como herramienta para el desarrollo de las actividades descritas anteriormente. Algunas de éstas técnicas de describen a continuación:

- Entrevistas: Generalmente van acompañadas de cuestionarios, donde el analista de sistemas interroga al cliente de manera abierta y sin presión. Es ideal al comienzo del proyecto ya que ayuda a dar una idea global del

problema, teniendo en cuenta el punto de vista de los usuarios involucrados, quienes pueden ser los gerentes, usuarios de sistemas antiguos y usuarios potenciales de la aplicación a desarrollar.

- Lluvia de ideas: Pretende que los involucrados en un proyecto desarrollen su creatividad y propongan la mayor cantidad de ideas posibles, de modo que se puedan analizar y llegar a elegir la óptima.
- Prototipos: Luego de la definición de los requerimientos, el equipo de desarrollo puede realizar modelos preliminares del producto software llamados prototipos, que a través de un diseño rápido y teniendo en cuenta los aspectos más globales de la definición de requisitos realizada, proveen alguna funcionalidad al cliente, el cual debe realizar un refinamiento de los requerimientos y realimentar el modelo propuesto por el prototipo. Éste es un proceso iterativo que puede darse hasta la entrega del proyecto donde los requerimientos son satisfechos a cabalidad.
- Casos de uso: Propone describir el sistema de acuerdo a las funciones que ofrece a su entorno, desde el punto de vista del usuario y de los servicios que ofrece a éste (teniendo en cuenta que el usuario no solo es una persona, sino que también puede ser otro sistema, o una máquina).
- Proceso de análisis jerárquico: Facilita la organización jerárquica de los requerimientos teniendo en cuenta su importancia y a identificar conflictos entre unos y otros requerimientos. Es apropiada cuando la cantidad de requerimientos que se maneja dentro de un proyecto no es muy grande.

Algunas de estas técnicas pueden utilizarse juntas, y puede buscarse la manera de combinarlas y coordinarlas de modo que favorezcan el proceso de especificación de requerimientos de un proyecto software de modo particular.

2.4. ESTÁNDAR IEEE 830 – PRÁCTICA RECOMENDADA PARA LA ESPECIFICACIÓN DE REQUERIMIENTOS DE SOFTWARE

Un estándar que enuncia una serie de recomendaciones para la especificación de requisitos de software es el Estándar IEEE 830 (IEEE Recommended Practice for Software Requirements Specifications), que permite realizar una especificación de calidad, donde el resultado obtenido del proceso de especificación de requerimientos de software, es un documento de especificación que pretende ser completo y sin ambigüedad.

Aunque éste estándar es muy general, se puede adaptar y redefinir para ser utilizado de modo más ajustado a las necesidades particulares de una organización, pero no se considera apropiado para quienes utilizan como metodología el desarrollo rápido de aplicaciones (DRA)

De acuerdo al estándar, una buena SRS debería proveer entre otros los siguientes beneficios:

- Establecer las bases de acuerdo de lo que tiene que hacer el software, entre el cliente y el proveedor que desarrolla el software.
- Reducir el esfuerzo de desarrollo, de modo que se haga una especificación rigurosa antes de empezar la fase de diseño del software.
- Proveer una base de estimación de costos y cronograma del proyecto.
- Proveer una base para la validación y verificación de requerimientos

En adelante se expondrán algunos aspectos que están contemplados en el estándar, para poder obtener una visión general de su contenido y de su uso en la SRS.

2.4.1. Definiciones

En general las definiciones de los términos usados en estas especificaciones están conforme a las definiciones proporcionadas en IEEE Std 610.12-1990.

- *Contrato*: Es un documento legal y en el estarán de acuerdo las partes del cliente y proveedor. Esto incluye los requisitos técnicos y requerimientos de la organización, costo y tiempo para un producto. Un contrato también puede contener la información informal pero útil como los compromisos o expectativas de las partes involucradas.
- *Cliente*: Es la persona que pagan por el producto y normalmente (pero no necesariamente) definen los requisitos. En la práctica el cliente y el proveedor pueden ser miembros de la misma organización.
- *Proveedor*: La persona que producen un producto para un cliente.
- *Usuario*: Es la persona que opera o actúa recíproca y directamente con el producto. El usuario y el cliente no son a menudo la misma persona.

2.4.2. Consideraciones para producir una buena SRS.

Estas cláusulas proporcionan información a fondo que deben ser consideradas al momento de producir una SRS. Esto incluye lo siguiente:

2.4.2.1. La Naturaleza de la SRS

La SRS son especificaciones para un producto software en particular, programa, o juego de programas que realizan ciertas funciones en un ambiente específico. La SRS puede escribirse por uno o más representantes del proveedor, uno o más representantes del cliente, o por ambos(recomendado).

Los problemas básicos que se presentan al escribir una SRS van dirigidos a lo siguiente:

a) La Funcionalidad. ¿Qué se supone va hacer el software?
b) Las interfaces Externas. ¿Cómo el software interactúa con las personas, el hardware del sistema, otro hardware, y otro software?
c) El Rendimiento. ¿Cuál es la velocidad, la disponibilidad, tiempo de respuesta, tiempo de recuperación de varias funciones del software, etc.?
d) Los Atributos. ¿Cuáles son las consideraciones de portabilidad, exactitud, el mantenimiento, seguridad, etc.?
e) Las restricciones del diseño impuestas para la implementación. ¿Hay algún requerimiento Standard, idioma de aplicación, políticas para la integridad del banco de datos, límites de los recursos, ambiente de operación, etc.?

2.4.2.2. El Ambiente de la SRS

Es importante considerar el papel que juega la SRS en el diseño del proyecto total, el cual se define en IEEE Std 610.12-1990. El software puede contener toda la funcionalidad del proyecto esencialmente o puede ser parte de un sistema más grande. En el último caso habrá una SRS que declarará las interfaces entre el sistema y su software modular, y la funcionalidad entre las dos. Desde que la SRS tiene un papel específico en el proceso de desarrollo de software, el que define la SRS debe tener el cuidado para no ir más allá de los límites de ese papel. Esto significa que:

a) Debe definir todos los requisitos del software correctamente.
b) No debe describir ni el diseño ni los detalles de la implementación. Éstos deben describirse en la fase de diseño del proyecto.
c) No debe imponer restricciones adicionales al software. Éstas se especifican propiamente en otros documentos.

2.4.2.3. Las Características de una buena SRS

✓ **Correcta**

Una SRS es correcta si, y sólo si, cada requisito declarado se encuentra en el software. No hay ninguna herramienta o procedimiento que aseguren la exactitud. Alternativamente el cliente o el usuario pueden determinar si la SRS refleja las necesidades reales correctamente. Identificando los requerimientos hace este procedimiento más fácil y hay menos probabilidad al error.

✓ **Inequívoco (No-Ambiguo)**

Una SRS es inequívoca si, y sólo si, cada requisito declarado tiene sólo una interpretación. Como mínimo, se requiere que cada característica de la última versión del producto se describa usando un único término. En casos dónde un término en un contexto particular tenga significados múltiples, el término debe ser incluido en un glosario dónde su significado tiene un sentido más específico.

La SRS es una parte importante del proceso de requisitos del ciclo de vida del software y se usa en el diseño, aplicación, supervisión, comprobación, aprobación y pruebas como está descrito en IEEE Std 1074-1997. El SRS debe ser inequívoco para aquéllos que lo crean y para aquéllos que lo usan. Sin embargo, estos grupos no tienen a menudo el mismo fondo y por consiguiente no tienden a describir los requisitos del software de la misma manera.

✓ **Completo**

Una SRS está completa si, y sólo si, incluye los siguientes elementos:

a) Todos los requisitos significativos, relacionando la funcionalidad, el desarrollo, las restricciones del diseño, los atributos y las interfaces externas. En particular debe reconocerse cualquier requisito externo impuesto por una especificación del sistema y debe tratarse.
b) La definición de las respuestas del software a todos los posibles datos de la entrada del sistema y a toda clase de situaciones. Una nota que es importante especificar son las contestaciones a las entradas válidas e inválidas a ciertos valores.
c) Tener todas las etiquetas y referencias a todas las figuras, tablas, diagramas en la SRS y definición de todas las condiciones y unidades de medida.

✓ **Consistente**

La consistencia se refiere a la consistencia interna. Si una SRS no está de acuerdo con algún documento del nivel-superior, como una especificación de requisitos de sistema, entonces no es correcto. Una SRS es internamente consistente si, y sólo si, ningún subconjunto de requisitos individuales genera conflicto.

✓ **Clasificado por importancia y/o estabilidad**

Una SRS está clasificada por importancia y/o estabilidad si cada requisito en ella tiene un identificador para indicar la importancia o estabilidad de ese requisito en particular. Generalmente, todos los requisitos que relacionan a un producto software no son igualmente importantes. Algunos requisitos pueden ser esenciales, sobre todo para las aplicaciones de vida crítica, mientras otros pueden ser deseables.

Cada requerimiento en la SRS debe identificarse para representar estas diferencias, aclarar y ser explícito. Esto ocasiona que los clientes den las consideraciones muy cuidadosamente a cada requisito para que se clarifique cualquier omisión que ellos pueden tener y que los diseñadores hagan diseños correctos y pongan el mismo esfuerzo en todos los niveles del producto del software.

✓ **Verificable**

Una SRS es verificable si, y sólo si, cada requisito declarado es verificable. Un requisito es verificable si, y sólo si, allí existe algún proceso por el cual una persona o máquina puede comprobar que el producto software tiene el requisito. En general cualquier requisito ambiguo no es verificable.

Los requisitos que incluyen frases como "trabaja bien", "interface humana buena" y "normalmente pasará", no pueden verificarse porque es imposible definir las condiciones "bueno," "bien" o "normalmente". Un ejemplo de una declaración verificable es: La salida del programa se producirá dentro de 20 segundos del evento el 60% del tiempo; y se producirá dentro de 30 segundos del evento el 100% del tiempo. Esta declaración puede verificarse porque usa condiciones concretas y cantidades mensurables.

✓ **Modificable**

Una SRS es modificable si, y sólo si, su estructura y estilo son tales que puede hacerse cualquier cambio a los requisitos de forma fácil, completa y consistentemente mientras conserva la estructura y estilo.

La redundancia no es un error, pero puede llevar fácilmente a los errores. La redundancia puede ayudar a hacer una SRS más legible de vez en cuando, pero un problema puede generarse cuando el documento redundante se actualiza. Por ejemplo, un requisito puede alterarse en un solo lugar dónde aparece. Entonces, la SRS se pone incoherente. Siempre que la redundancia sea necesaria, la SRS debe incluir referencias cruzadas para hacerlo modificable.

✓ **Trazable**

Una SRS es trazable si el origen de cada uno de sus requisitos está claro y si facilita las referencias de cada requisito en el desarrollo futuro o documentación del mismo.

2.4.2.4. Preparación Conjunta de la SRS

El proceso de desarrollo de software debe empezar con un acuerdo entre el proveedor y el cliente en lo que el software debe hacer. Este acuerdo, en la forma de una SRS, debe ser preparada conjuntamente. Esto es importante porque ni el cliente ni el proveedor son calificables para escribir exclusivamente una buena SRS solos. Por consiguiente, el cliente y el proveedor deben trabajar juntos para producir una SRS completa y comprensible.

2.4.2.5. La evolución de la SRS

La SRS puede necesitar evolucionar en la medida en que avance el desarrollo del producto software. Puede ser imposible especificar algunos detalles en el momento que el proyecto se inicia. Los cambios adicionales pueden ocurrir de acuerdo a como se vayan descubriendo las deficiencias, las limitaciones e inexactitudes en la SRS.

2.4.2.6. Prototipos

Los prototipos frecuentemente se usan durante una fase de los requisitos de un proyecto. Existen muchas herramientas para generar un prototipo que exhiba algunas características de un sistema, creado rápida y fácilmente. Los prototipos son útiles porque permiten desplegar algunos aspectos del sistema, producir respuestas y generar nuevas preguntas. Una SRS basada en un prototipo tiende a sufrir menos cambios durante el desarrollo.

Un prototipo debería usarse como una forma de definir los requisitos del software. Pueden extraerse algunas características como pantalla o formatos de reporte directamente del prototipo. Otros requisitos pueden ser inferidos ejecutando experimentos con el prototipo.

2.4.2.7. Generando el diseño en la SRS

Un requisito especifica cualquier función externa visible o atributo de un sistema. Un diseño describe un subcomponente particular de un sistema y/o sus interfaces con otros subcomponentes. El diseñador de la SRS debe distinguir claramente entre identificar las restricciones del diseño requeridos y proyectar un diseño específico. Cada requisito en la SRS limita las alternativas del diseño. Esto no significa, sin embargo, que cada requisito es el diseño. La SRS debe especificar qué funciones serán realizadas, con qué datos, para producir qué resultados, en qué situación y para quien. La SRS se debe enfocar en los servicios que serán realizados.

2.4.2.8. Generando los requisitos del proyecto en la SRS.

La SRS debe dirigir el producto software, no el proceso de producir el producto software. Los requisitos del proyecto representan un acuerdo entre el cliente y el proveedor de tipo contractual, que pertenecen a la producción de software y no deben ser incluidos en el SRS. Éstos requisitos normalmente incluyen puntos como:

- Costo
- Tiempos de la entrega
- Reporte de procedimientos

- Métodos de desarrollo de Software
- Aseguramiento de la Calidad
- Criterios de validación y aprobación
- Procedimientos de aceptación

2.4.3. Las partes de una SRS

A continuación se describe una estructura para realizar una SRS. Una SRS no tiene que seguir este modelo o usar los nombres dados aquí para sus partes, pero sí una buena SRS debe incluir toda la información que se describe aquí.

Tabla de Contenido
1. Introducción
1.1 Propósito
1.2 Alcance
1.3 Definiciones, siglas, y abreviaciones
1.4 Referencias
1.5 Apreciación global
2. Descripción global
2.1 Perspectiva del producto
2.2 Funciones del producto
2.3 Características del usuario
2.4 Restricciones
2.5 Suposiciones y dependencias
3. Requisitos específicos
Apéndices
Índice

Figura 2. Estructura de la SRS

2.4.3.1. Introducción

La introducción de la SRS debe proporcionar una apreciación global de la SRS completa. Debe contener las siguientes subdivisiones:

2.4.3.1.1. Propósito

Esta subdivisión debe:
a) Delimitar el propósito de la SRS
b) Especificar a que público va dirigida la SRS

2.4.3.1.2. Alcance

Esta subdivisión debe:
a) Identificar por el nombre el(los) producto(s) software a desarrollar (por ejemplo, Host DBMS, Generador de Reportes, etc.)
b) Explicar lo que el(los) producto(s) software hará.

c) Describir la aplicación software especificándo los beneficios pertinentes, objetivos, y metas.
d) Ser consistente con las declaraciones similares en las especificaciones de alto nivel (por ejemplo, las especificaciones de los requisitos del sistema), si ellos existen.

2.4.3.1.3. Definiciones, siglas, y abreviaciones

Esta subdivisión debe proporcionar las definiciones de todas las condiciones, las siglas, y abreviaciones requeridas para interpretar la SRS apropiadamente. Esta información puede proporcionarse por la referencia a uno o más apéndices en la SRS o por la referencia a otros documentos.

2.4.3.1.4. Referencias

Esta subdivisión debe:
a) Proporcionar una lista completa de todos los documentos referenciados en cualquier parte de la SRS.
b) Identificar cada documento por título, número del reporte (si es aplicable), fecha de publicación y editorial.
c) Especificar las fuentes de las referencias de donde se obtuvieron.
Esta información puede proporcionarse por la referencia a un apéndice o a otro documento.

2.4.3.1.5. Apreciación global

Esta subdivisión debe:
a) Describir lo que contiene el resto de la SRS
b) Explicar cómo está organizada la SRS

2.4.3.2. Descripción global

Esta sección de la SRS debe describir los factores generales que afectan el producto y sus requisitos. Esta sección no declara los requisitos específicos. En cambio, provee un fondo para esos requisitos que se definen en detalle en la Sección 3 de la SRS y los hacen más fácil entender. Esta sección normalmente consiste en seis subdivisiones, como sigue:

2.4.3.2.1. Perspectiva del producto

Esta subdivisión de la SRS debe poner el producto en perspectiva con otros productos relacionados. Si el producto es independiente y totalmente autónomo, debe declararse que así es. Si la SRS define un producto que es un componente de un sistema más grande, como frecuentemente ocurre, entonces esta subdivisión debe relacionar los requisitos de ese sistema más grande a la funcionalidad del software y debe identificar las interfaces entre ese sistema y el software. Un diagrama del bloque que muestra los

componentes mayores del sistema más grande, las interconexiones, y las interfaces externas pueden ser útiles.

Esta subdivisión también debe describir cómo el software opera dentro de varias restricciones. Por ejemplo, estas restricciones podrían incluir:

a) Interfaces del sistema. Esto debe listar cada interfaz del sistema y debe identificar la funcionalidad del software para satisfacer el requisito del sistema y la descripción de la interfaz para empatar el sistema.

b) Interfaces con el usuario. Esto debe especificar a lo siguiente:
a) Las características lógicas de cada interfaz entre el producto software y sus usuarios. Esto incluye las características de la configuración (por ejemplo, formatos de la pantalla requeridos, página o esquemas de la ventana, los reportes o menús, entre otros) necesaria para satisfacer los requisitos del software.
b) Todos los aspectos para optimizar la interfaz con la persona que debe usar el sistema. Esto puede comprender una lista de lo que debe y no aparecer hacer el sistema para el usuario. Un ejemplo puede ser un requisito para la opción de mensajes de error largos o cortos. Como todos, estos requisitos deben ser verificables.

c) Interfaces de hardware. Esto debe especificar las características lógicas de cada interfaz entre el producto software y los componentes del hardware del sistema. Esto incluye las características de la configuración (el número de puertos, conjuntos de instrucción, etc.), también cubre qué dispositivos serán soportados, cómo ellos serán soportados y protocolos.

d) Interfaces de software. Esto debe especificar el uso de otros productos software requeridos e interfaces con otros sistemas Para cada producto del software requerido debe proporcionarse:
- El nombre;
- El código mnemotécnico;
- El número de la especificación;
- El número de la versión;
- La fuente.

Para cada interfaz, lo siguiente debe proporcionarse:
- La discusión del propósito de la interfaz del software en relación con el producto del software.
- La definición de la interfaz por lo que se refiere a los mensajes contenidos y formatos. No es necesario detallar cualquiera bien la documentación de la interfaz, pero una referencia al documento que define la interfaz se requiere.

e) Interfaces de comunicaciones. Esto debe especificar las interfaces para comunicaciones como protocolos de redes locales, etc.,

f) Restricciones de memoria. Esto debe especificar cualquier característica aplicable y límites en la memoria primaria y secundaria.

g) Funcionamiento. Esto debe especificar el modo de funcionamiento normal y especial requerido por el usuario como:
a) Los varios modos de funcionamiento del usuario en la organización
b) Los Periodos de funcionamientos activos e inactivos;
c) Datos que procesan las funciones de apoyo;
d) Operaciones de recuperación y copia de seguridad.
NOTA - Esto a veces se especifica como parte de la sección de interfaces de usuario.

h) Requisitos de adaptación.
Esto debe:
a) Definir los requisitos para cualquier dato o secuencia de inicialización que son específicos a un sitio dado, misión o modo de operación
b) Especifique el sitio o los rasgos que se deben relacionar que deben modificarse para adaptar el software a una instalación particular.

2.4.3.2.2. Funciones del Producto
Esta subdivisión de la SRS debe proporcionar un resumen de las funciones principales que el software realizará. A veces el resumen de la función que es necesario para esta parte puede tomarse directamente de la sección de especificacines de alto nivel (si existe) eso asigna las funciones particulares al producto del software.

2.4.3.2.3. Características del usuario
Esta subdivisión de la SRS debe describir esas características generales de los usuarios del producto, que incluye nivel de escolaridad, experiencia, y formación técnica.

2.4.3.2.4. Restricciones
Esta subdivisión de la SRS debe proporcionar una descripción general de cualquier otro punto que limitará las opciones de los diseñadores. Esto incluye:
a) políticas reguladoras
b) limitaciones de Hardware
c) Interfaces con otras aplicaciones
d) Operaciones en Paralelo
e) Funciones de Auditoria;
f) Funciones de Control;
g) Requisitos de lenguaje;

h) Protocolos
i) Requisitos de Fiabilidad;
j) Seguridad y consideraciones de garantía.

2.4.3.2.5. Suposiciones y dependencias

Esta subdivisión de la SRS debe listar cada uno de los factores que afectan los requisitos declarados en la SRS. Estos factores no son restricciones de diseño del software pero son, más bien, cualquier cambio que puede afectar los requisitos en la SRS. Por ejemplo, una suposición puede ser que un sistema operativo específico estará disponible para el hardware designado para el producto software. Si, de hecho, el sistema operativo no está disponible, la SRS tendría que cambiar de acuerdo a éste suceso..

2.4.3.3. Requisitos específicos

Esta sección de la SRS debe contener todos los requisitos del software a un nivel de detalle suficiente para permitirles a los diseñadores diseñar un sistema para satisfacer esos requisitos, y a los auditores probar que el sistema satisface esos requisitos. A lo largo de esta sección, cada requisito declarado debe ser externamente perceptible por los usuarios, operadores u otros sistemas externos. Estos requisitos deben incluir por lo menos una descripción de cada entrada (estímulo) en el sistema, cada salida (respuesta) del sistema, y todas las funciones realizadas por el sistema en respuesta a una salida o a una entrada o que apoye una salida. Esta es la parte más grande y más importante del SRS.

2.4.3.3.1. Interfaces externas

Ésta debe ser una descripción detallada de todas las entradas y salidas del sistema software. Debe complementar las descripciones de interfaces descritas y no debe repetirse la información allí. Debe estructurarse como sigue:
a) Nombre de item;
b) Descripción de propósito;
c) Fuente de entrada o destino de salida;
d) Rango válido, exactitud, y/o tolerancia;
e) Unidades de medida;
f) Tiempos;
g) Relaciones a otras entradas/salidas;
h) Formato de pantalla /organización;
i) Formato de ventanas/organización;
j) Formatos de los datos;
k) Formatos de los comandos;

2.4.3.3.2. Funciones

Los requisitos funcionales deben definir las acciones fundamentales que deben tener lugar en el software, en la aceptación y procesamiento de las entradas y las salidas. Éstos generalmente se listan como declaraciones "debe" que empiezan con "El sistema debe…." Éstos incluyen:

a) Verificar la validez sobre las entradas
b) Secuencia exacta de las operaciones
c) Respuestas a situaciones anormales, incluyendo desbordamiento, facilidades de comunicación y manejo de errores y recuperación
d) Efecto de los parámetros
e) Relación de salidas a las entradas, incluyendo secuencias de entrada/salidas y las fórmulas de entrada y su conversión a la salida.

2.4.3.3.3. Requisitos de desarrollo.

Esta subdivisión debe especificar los requerimientos estáticos y dinámicos que se pusieron en el software o en la interacción humana con el software en conjunto. Los requisitos estáticos pueden incluir a lo siguiente:

a) Número de terminales a ser soportadas;
b) Número de usuarios simultáneos a ser soportados;
c) Cantidad y tipo de información que se manejara.

A veces se identifican los requisitos estáticos bajo una sección separada titulada la Capacidad. Por ejemplo, los requisitos dinámicos pueden incluir los números de transacciones, tareas y la cantidad de datos a ser procesados dentro de ciertos periodos de tiempo para condiciones de trabajo normales y máximas. Todos que estos requisitos deben declararse en condiciones mensurables. Por ejemplo, 95% de las transacciones se procesarán en menos de 1 seg.

2.4.3.3.4. Requisitos lógicos de la base de datos

Esto debe especificar los requisitos lógicos para cualquier información que será puesta en la base de datos. Esto puede incluir a lo siguiente:

a) Tipos de información usadas por varias funciones;
b) Frecuencia de uso;
c) Capacidades de acceso;
d) Entidades de datos y sus relaciones;
e) Restricciones de integridad;
f) Requerimientos en la retención de datos.

2.4.3.3.5. Restricciones de diseño.

Esto debe especificar las restricciones de diseño derivadas por otros estándares, limitaciones de hardware, etc.

2.4.3.3.6. Atributos del software del sistema.

Hay varios atributos del software que puede servir como requisitos. Es importante que los atributos requeridos se especifiquen para que puedan verificarse objetivamente. Como ejemplo, se provee la siguiente lista:

a) Fiabilidad. Esto debe especificar los factores necesarios para establecer la fiabilidad requerida del sistema software al momento de la entrega.

b) Disponibilidad. Esto debe especificar los factores necesarios para garantizar un nivel de disponibilidad definido para el sistema.

c) Seguridad. Esto debe especificar los factores que protegen el software del acceso accidental o malévolo, uso, modificación, destrucción o descubrimiento. Los requisitos específicos en esta área podrían incluir la necesidad de que se utilice ciertas técnicas de encriptamiento, se tengan Logs de entrada o históricos de datos, se asigne ciertas funciones a módulos diferentes, se restrinja las comunicaciones entre algunas áreas del programa, la integridad de datos se verifique para variables críticas.

d) Mantenimiento. Esto debe especificar atributos de software que se relacionan a la facilidad de mantenimiento del propio software. Puede haber algún requisito para cierta modularidad, interfaces, complejidad, etc.

e) Portabilidad. Esto debe especificar atributos de software que se relacionan con la facilidad de poner el software a otro servidor y/o sistemas operativos.

2.4.3.3.7. Organizar los requisitos específicos.

Por trivial que parezca, el detalle de los requerimientos tiente a ser extenso. Por esta razón, se recomienda ser cuidadosos al organizar éstos requerimientos de una manera óptima para que sean entendibles. No existe una organización óptima para todos los sistemas. A continuación se describen algunas de esas formas de organizar la SRS.

a) Modo del sistema. Algunos sistemas se comportan diferentes dependiendo del modo de operación. Por ejemplo, un sistema de control puede tener conjuntos diferentes de funciones que dependen de su modo de operación: entrenando, normal o emergencia.

b) Clases de usuario. Algunos sistemas proporcionan diferentes grupos de funciones a diferentes clases de usuarios. Por ejemplo, un sistema de mando de ascensor presenta las capacidades diferentes a los pasajeros, obreros de mantenimiento y bomberos.

c) Objetos. Los objetos son entidades del mundo real que tienen una contraparte dentro del sistema. Por ejemplo, en un sistema que supervisa pacientes, los objetos incluyen pacientes, sensores, enfermeras, cuartos, médicos, medicinas, etc. Asociado con cada objeto hay un grupo de atributos y funciones.

d) Rasgo. Un rasgo es un servicio deseado externamente por el sistema que puede exigir una secuencia de entradas para efectuar el resultado deseado. Por ejemplo, en un sistema del teléfono, los rasgos incluyen la llamada local, llamada remitida y llamada en conferencia. Cada rasgo generalmente se describe en una secuencia de estímulo-respuesta.

e) Estímulo. Algunos sistemas pueden organizarse mejor describiendo sus funciones en relación a los estímulos. Por ejemplo, pueden organizarse las funciones del sistema de aterrizaje de un avión automático en secciones para la pérdida del control, esquivación del viento, el cambio súbito en el destino, la velocidad vertical excesiva, etc.

f) Contestación. Algunos sistemas pueden organizarse mejor describiendo todas las funciones en apoyo a la generación de respuestas. Por ejemplo, pueden organizarse las funciones de un sistema de personal en secciones que corresponden a todas las funciones asociadas con los sueldos generados, con generar una lista actual de empleados, etc.

g) Jerarquía Funcional. Cuando ninguno de los esquemas orgánicos anteriores demuestra ser útil, la funcionalidad global puede organizarse en una jerarquía de funciones organizada por cualesquier entradas comunes, salidas comunes o el acceso común de los datos internos. Los diagramas de flujo y diccionarios de datos pueden usarse para mostrar las relaciones entre las funciones y datos.

2.4.3.4. Información de apoyo

La información de apoyo hace más fácil el uso de la SRS. Incluye lo siguiente:

2.4.3.4.1. Tabla de contenidos e índice

La tabla de contenidos e índice es bastante importante y debe seguir las prácticas de composiciones generales.

2.4.3.4.2. Apéndices

Los apéndices no siempre son considerados parte de la SRS real y no siempre son necesarios. Ellos pueden incluir:

a) Ejemplos de formatos de las entradas/salidas, descripciones del análisis del costo o resultados de estudios del usuario;
b) Información a fondo que puede ayudar a los lectores del SRS;
c) Una descripción de los problemas a ser resueltos por el software;

Cuando los apéndices son incluidos, la SRS debe declarar explícitamente si los apéndices serán considerados parte de los requisitos o no.

Para este proyecto se decidió hacer una agrupación del estándar para obtener una estructura de la siguiente manera: Primero, se clasificaron como atributos descriptores del proyecto la Introducción y Descripción Global con sus ítems (a atributos) correspondientes y se dejó como pieza principal la parte de Requisitos Específicos con sus atributos respectivos; es decir, la estructura tomó la siguiente forma:

I. ATRIBUTOS DEL PROYECTO
I.1. Introducción
I.1.1 Propósito
I.1.2 Alcance
I.1.3 Definiciones, siglas, y abreviaciones
I.1.4 Referencias
I.1.5 Apreciación global
I.2. Descripción global
I.2.1 Perspectiva del producto
I.2.2 Funciones del producto
I.2.3 Características del usuario
I.2.4 Restricciones
I.2.5 Suposiciones y dependencias
II. REQUERIMIENTOS DEL PROYECTO
II.1. Requisitos específicos

Figura 2a. Estructura Adaptada de la SRS

3. MARCO CONCEPTUAL

En éste capítulo se describen algunos aspectos técnicos aplicados para el desarrollo del prototipo PASCAR, la plataforma utilizada y herramientas de desarrollo que se emplearon. No es un capítulo que ahonde en aspectos específicos, si no que se escribió con la intención de dar una idea general de las tecnologías involucradas.

3.1. PLATAFORMA Y DISPOSITIVO UTILIZADO

El dispositivo móvil elegido para el proyecto fue Pocket PC, un dispositivo de bolsillo similar a un computador de escritorio, pero de dimensiones reducidas y capacidades limitadas. Los Pocket PC trabajan bajo sistema operativo Windows especiales (Windows CE, Windows Mobile) que le permiten administrar aplicaciones personales y profesionales, casi como en un ordenador de escritorio.

Windows Mobile es un sistema operativo compacto diseñado para dispositivos móviles (celulares, PDA's) similar a las versiones de escritorio de Windows de Microsoft, que contiene prestaciones multimedia, acceso remoto a datos e Internet. Para el proyecto fue utilizada la Segunda Edición de Windows Mobile 2003 para Pocket PC.

Para efectos de desarrollo se requirió el uso de un emulador para realizar las pruebas y ejecutar la aplicación móvil sin necesidad de un dispositivo físico. Se utilizó una aplicación de escritorio que emula el comportamiento de un dispositivo móvil basado en la plataforma Windows CE o Windows Mobile llamado Microsoft Device Emulator v1.0.

3.2. HERRAMIENTAS DE DESARROLLO

3.2.1. Aplicación Móvil

Algunas de las herramientas disponibles para el desarrollo de aplicaciones en Pocket PC son: J2ME (java), Waba y SuperWaba (compatibles con java), AppForge (Visual Basic) y Satellite Forms (Visual Basic Script). Para la selección de la herramienta de desarrollo de la aplicación móvil se tuvo en cuenta que la herramienta fuera de fácil aprendizaje en poco tiempo, que guardara cierta familiaridad con los lenguajes utilizados y que tuviera una interfaz gráfica que facilitara el diseño. Se eligió trabajar con Satellite Forms que es una herramienta de desarrollo que permite crear aplicaciones para

dispositivos Pocket PC (2002 y 2003) y Palm OS, utilizando lenguaje script para Visual Basic que resultó bastante familiar.

3.2.2. Aplicación de Escritorio

En cuanto a la herramienta seleccionada para el desarrollo de la aplicación de escritorio se eligió Visual Basic 6.0 de Microsoft, que facilita el desarrollo de aplicaciones con interfaces gráficas agradables al usuario a través de las referencias y controles que provee. Además facilita la conexión a bases de datos, como MySQL que fue utilizado como el manejador de base de datos para la aplicación de escritorio.

4. DISEÑO E IMPLEMENTACIÓN DEL PROTOTIPO

4.1. METODOLOGÍA APLICADA

Para seleccionar la metodología de desarrollo de software para el proyecto, se analizaron los siguientes modelos: espiral, cascadas modificadas, prototipado evolutivo, entrega por etapas y entrega evolutiva. Éste último modelo fue seleccionado como la metodología a utilizar en el desarrollo del proyecto por las siguientes razones:

- La entrega evolutiva proporciona la flexibilidad del prototipado evolutivo y las características de planificación de la entrega por etapas. Es decir, se planifica el desarrollo de versiones durante el proyecto, en las cuales se elabora un prototipo del software, que es estregado al cliente para su realimentación. Esto le brinda cierta funcionalidad al cliente antes que el prototipo sea entregado como producto final del proyecto. Además permite refinar los requerimientos del software, que pueden no haber sido definidos detalladamente al inicio del proyecto. Cuando el prototipo ha sido refinado completamente, se entrega como la versión final del proyecto.
- Cuando se hace una inclinación mayor hacia la entrega por etapas, se puede conocer qué es lo que se va a construir al comienzo del proyecto, sin tener que ahondar en niveles detallados de especificación, permitiendo cierta planificación del cronograma y presupuesto del proyecto.
- Según Steve McConnell[7], el modelo de entrega evolutiva se representa de la siguiente manera:

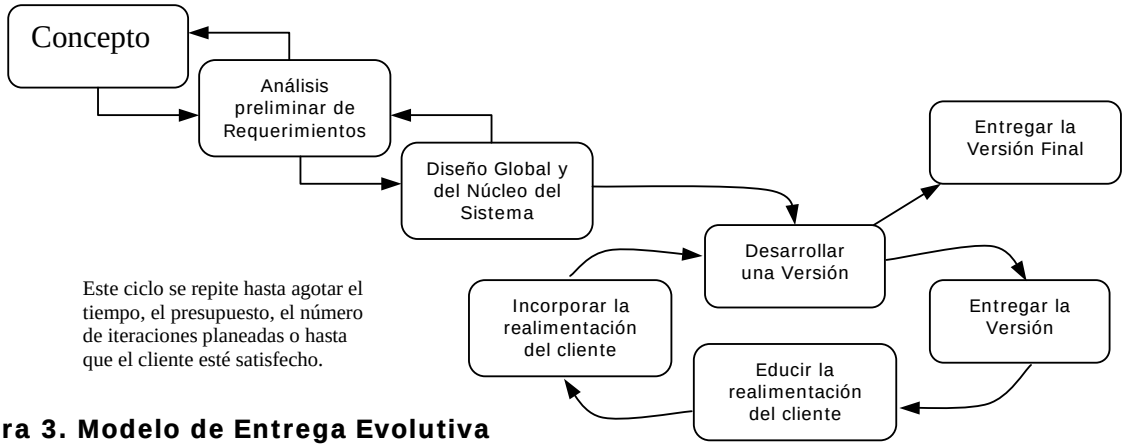


Figura 3. Modelo de Entrega Evolutiva

[7] McCONNELL, Steve. Desarrollo y Gestión de Proyectos Informáticos, Capítulo 7: Planificación del Ciclo de Vida, página 164. McGraw Hill, España, 1997.

4.2. FUNCIONALIDADES DE LA APLICACIÓN

El producto desarrollado en este proyecto es un prototipo software para la captura y administración de requerimientos de software diseñado para reforzar las tareas de especificación de requisitos de un proyecto software.

El proyecto PASCAR cuenta con dos aplicaciones: La principal, que es la aplicación de escritorio, y la secundaria que se ha dado como valor agregado al proyecto y es la aplicación móvil. La primera cuenta con toda la independencia funcional necesaria de modo que no requiera ninguna otra aplicación para trabajar de manera completa de acuerdo a los objetivos propuestos para la especificación y administración de requerimientos. La segunda, como valor agregado, es una herramienta adicional que es suministrada para permitir mayor movilidad al usuario de la aplicación de escritorio utilizando un PDA, y permite la creación y actualización de requerimientos desde el dispositivo para después ser sincronizado con la aplicación de escritorio.

Este prototipo cuenta con las siguientes funcionalidades generales:

- ***Autenticación de usuarios.*** Se utiliza para controlar el acceso a la aplicación (tanto la móvil como la de escritorio), de los usuarios registrados.

- ***Gestión de Proyectos de Software***. A través de éste módulo se podrán registrar los datos preliminares de un proyecto software. Consta de dos módulos: uno para la administración de proyectos donde se crea, consulta o elimina cada proyecto con su información básica; el otro módulo, para la especificación de proyectos, donde se registran aspectos generales del proyecto y requerimientos de manera poco detallada.

- ***Gestión de Requerimientos de Software***. Es el núcleo de la aplicación. Es el módulo en el que se registrarán y actualizarán los requerimientos de cada proyecto, estructurados de acuerdo a una clasificación por tipos de requerimientos (req. Funcionales, de diseño, entre otros)

- ***Generación de Documentación de Requerimientos***. Se utiliza para generar los documentos de especificación de requerimientos de software de cada proyecto.

- ***Sincronización de datos entre las dos aplicaciones.*** Es un módulo dedicado a la sincronización de datos entre las aplicaciones del dispositivo

móvil y el computador de escritorio y realizar las tareas de actualización de requisitos.

- ***Mantenimiento del sistema.*** Es el módulo utilizado para la administración de usuarios de la aplicación y la administración de los dispositivos móviles que tendrán acceso a la aplicación. Además se podrán adicionar y administrar los tipos de requerimientos y atributos que puedan ser utilizados para la especificación de un proyecto.

4.3. ACTORES DE LA APLICACIÓN

- ACTORES PASCAR PARA EL PROTOTIPO DE ESCRITORIO

a) *Actor Administrador:* Es el usuario con acceso a todas las funcionalidades de la aplicación de escritorio, quien administra usuarios, dispositivos y hace mantenimiento de la base de datos. Éste actor puede ser un analista de sistemas.

b) *Actor Analista de Sistemas / Ingeniero de Requerimientos*: Es el usuario encargado de administrar los proyectos software a su mando. Tendrá a su disposición gran parte de las funcionalidades de la aplicación.

c) *Actor Diseñador / Desarrollador de software*: Es un usuario secundario de la aplicación, que podrá consultar la información y documentación relacionada con los proyectos, pero solo podrá hacer sugerencias y recomendaciones acerca de los cambios que deben realizarse en la especificación de los requerimientos de software.

- ACTORES PASCAR PARA EL PROTOTIPO MÓVIL

a) *Actor Analista de Sistemas / Ingeniero de Requerimientos*: Es el único usuario de la aplicación móvil. Tendrá a su disposición todas las funcionalidades de la aplicación móvil.

4.4. CASOS DE USO

De acuerdo a los módulos funcionales expuestos anteriormente y a los actores definidos, se describen los casos de uso propuestos para el prototipo PASCAR.

4.4.1. Autenticación de usuarios

4.4.1.1. CASO DE USO Autenticación de Usuarios

Figura 4. Caso de Uso Autenticación de Usuarios

- DESCRIPCIÓN

  Este caso de uso describe el acceso a la aplicación que pueden tener el administrador del sistema, el analista de sistemas (o ingeniero de requerimientos) y el ingeniero de desarrollo o diseñador que se encuentren registrados y activos en el sistema. El acceso en la aplicación móvil está restringido para uso exclusivo del analista de sistemas.

- RESTRICCIONES
  - El usuario debe estar registrado y activo en el sistema.
  - Para el acceso a la aplicación móvil, el usuario debe tener asignado y registrado un dispositivo móvil o PDA.

- PROCESO
  1. El usuario Ingresa al sistema su login (nombre de usuario) y su password (contraseña).
  2. El usuario hace clic en el botón Aceptar.
  3. El sistema debe validar que no se encuentren vacíos los campos de login y password.
  4. El sistema debe verificar que el login y password de usuario sean válidos, es decir, que se encuentren registrados en la base de datos del sistema.
  5. Si el usuario se encuentra registrado en el sistema, se permitirá el ingreso a la aplicación, con las funcionalidades disponibles de acuerdo al tipo de usuario que haya ingresado. Si el usuario no coincide con ninguno de los usuarios registrados, se emitirá un mensaje de error que indique el no acceso a la aplicación.

- ENTRADAS DEL PROCESO
  - Nombre de usuario (login) y Contraseña (password)

- SALIDAS DEL PROCESO
  - Entorno de trabajo de la aplicación disponible para el usuario autenticado.
  - Mensajes de error del sistema cuando la identificación del usuario no sea válida.

4.4.2. Gestión de proyectos de Software

Figura 5. Caso de Uso Gestión de Proyectos de Software

4.4.2.1. CASO DE USO Administración de Proyectos

- DESCRIPCIÓN

  A través de éste caso de uso, se podrán crear nuevos proyectos para la captura y administración de requerimientos de software. Se registrarán los datos más generales del proyecto, y en caso que el proyecto exista, se podrán modificar estos datos. Esta tarea será realizada únicamente por el analista de sistemas.

- RESTRICCIONES
  - No pueden registrarse dos proyectos con el mismo nombre y/o código de identificación.

- CONDICIONES

  Los datos requeridos para realizar un registro preliminar o la modificación de un proyecto son los siguientes:
  - Código de identificación del proyecto
  - Nombre del proyecto
  - Fecha de creación del proyecto
  - Hora de creación del proyecto
  - Identificación del cliente del proyecto software. Por tanto el cliente debe estar creado previamente en el sistema.

- PROCESO
    1. El usuario selecciona la opción Crear Proyecto.
    2. El sistema despliega una interfaz para el ingreso de la información requerida para un proyecto nuevo.
    3. El sistema debe validar que no se encuentren vacíos los campos que se consideren obligatorios.
    4. Luego de ingresar la información, el usuario hace clic en la opción Guardar.
    5. Si la información ha sido ingresada correctamente, se almacena la información en la base de datos.

    NOTA: El procedimiento a seguir para la modificación de los datos de un proyecto sigue la misma estructura y validaciones que para el caso de creación descrito anteriormente.

- ENTRADAS DEL PROCESO

    Todas las entradas del proceso están dadas por las solicitadas en el formulario desplegado por el sistema.

- SALIDAS DEL PROCESO

    La salida que se produce está directamente ligada con la inserción o actualización de los datos en la tabla correspondiente en la base de datos.

    En caso que los datos no sean ingresados correctamente, se emitirán los respectivos mensajes de error.

4.4.2.2. CASO DE USO Especificación de Proyectos

- DESCRIPCIÓN

    Se podrán registrar y consultar aspectos más específicos del proyecto y que no se han incluido en la especificación anterior (3.2.1.1)

- RESTRICCIONES
    - No se pueden registrar o actualizar los aspectos específicos de un proyecto si antes no se ha creado un proyecto para el cual definirlos.

- CONDICIONES

    Los datos requeridos para realizar un registro más detallado de un proyecto son los siguientes:

- Propósito
- Audiencia involucrada
- Alcances
- Definiciones, acrónimos y abreviaturas
- Referencias
- Resumen
- Perspectivas del producto
- Funciones del producto
- Características de los usuarios
- Restricciones generales
- Postulados y dependencias

- PROCESO
  - El usuario selecciona la opción para especificar los atributos del Proyecto.
  - El sistema despliega una interfaz para el ingreso de la información requerida para la especificación de un proyecto existente.
  - El sistema debe validar que no se encuentren vacíos los campos que se consideren obligatorios.
  - Luego de ingresar la información, el usuario hace clic en la opción Guardar.
  - Si la información ha sido ingresada correctamente y no se encuentran datos duplicados en el sistema, se almacena la información en la base de datos.
  - En caso contrario se emite el correspondiente mensaje de error.

- ENTRADAS DEL PROCESO

  Todas las entradas del proceso están dadas por las solicitadas en el formulario desplegado por el sistema.

- SALIDAS DEL PROCESO

  La salida que se produce está directamente ligada con la inserción de los datos en la tabla correspondiente en la base de datos.

  En caso que los datos no sean ingresados correctamente, se emitirán los respectivos mensajes de error.

4.4.3. Gestión de Requerimientos de Software

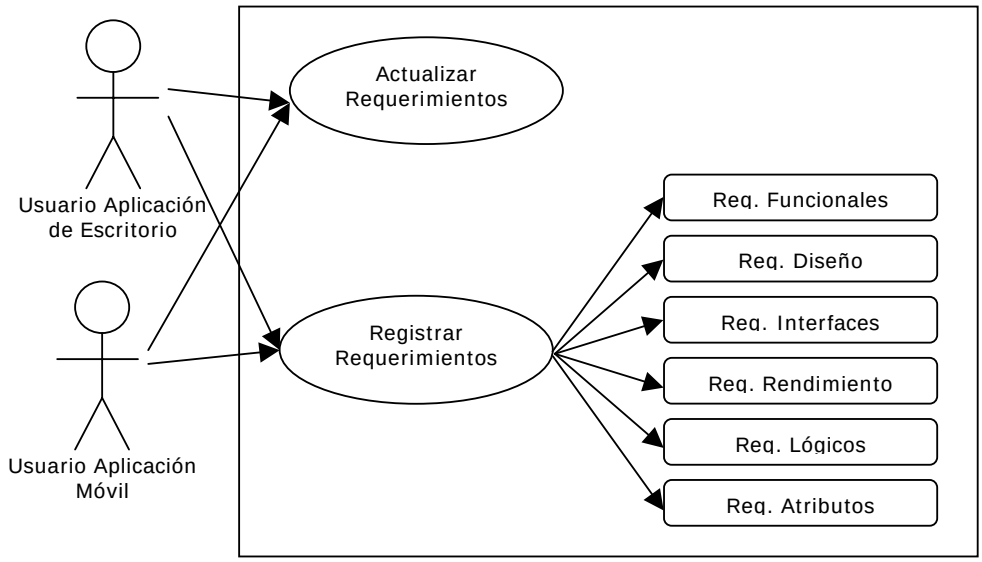


Figura 6. Caso de Uso Gestión de Requerimientos

4.4.3.1. CASO DE USO Registrar Requerimientos

- DESCRIPCIÓN

  Permite registrar los diferentes requerimientos que hacen parte de la SRS de un proyecto. Estos requerimientos se clasifican de la siguiente manera:

  - Requerimientos funcionales: Hace referencia a los requisitos que describen la funcionalidad de una aplicación, en función de sus entradas, procesos y salidas.
  - Requerimientos de rendimiento: Describe el rendimiento que se espera del sistema en productivo, como número de usuarios soportados simultáneamente, número de terminales, número de transacciones por unidad de tiempo, entre otros.
  - Requerimientos de restricciones de diseño: Hace referencia a las restricciones que se dan en la elaboración de un proyecto software debida a estándares o políticas internas de las organizaciones (proveedor y/o cliente) y a limitaciones en cuanto hardware y software.
  - Requerimientos de interfaces externas: Describe las interfaces de hardware, software y usuario (formatos de pantalla, contenidos de reportes, menús) y la comunicación entre éstas.

- Requerimientos de atributos de la aplicación: En esta parte se describen los atributos de seguridad, mantenibilidad de la aplicación, portabilidad y disponibilidad de la misma.
- Requerimientos lógicos: Hace referencia a la especificación de los tipos de datos utilizados, la frecuencia de uso de los datos, el acceso a los mismos y la integridad que deben manejar los datos.
- Otros requerimientos: Son los que no se encuentren dentro de la clasificación antes listada.

- RESTRICCIONES

  Las tablas de soporte donde se definen los tipos de requerimientos y sus atributos no deben encontrarse vacías. Para el caso de un requerimiento que no se encuentre dentro de la clasificación definida, se debe ingresar por el módulo de administración para requerimientos y crearlo.

- PROCESO
  - El usuario ingresa a un formulario que contiene cada una de las clasificaciones de tipos de requerimientos mencionadas anteriormente.
  - Cada tipo de requerimiento posee unos atributos propios que se deben registrar manualmente por parte del usuario.
  - Cuando se haya terminado la especificación de un requerimiento, el usuario debe hacer clic en la opción guardar, para salvar el requerimiento dentro del proyecto.
  - El sistema debe validar los campos que se encuentren vacíos y se consideren obligatorios.
  - Si la información está completa, los datos se deben almacenar en la tabla de requerimientos.

- ENTRADAS DEL PROCESO

  Las entradas del proceso están dadas por el formulario para el ingreso de los requerimientos de un proyecto, algunos de los cuales serán seleccionados por el usuario (corresponden a datos tomados de la base de datos para definir el tipo de requerimiento y sus atributos) y otros ingresados manualmente.

- SALIDAS DEL PROCESO

  La salida que se produce está directamente ligada con la inserción de los datos en la tabla correspondiente en la base de datos.

  En caso que los datos no sean ingresados correctamente, se emitirán los respectivos mensajes de error.

4.4.3.2. CASO DE USO Actualizar Requerimientos

- DESCRIPCIÓN

  Se podrán actualizar los diferentes requerimientos que hagan parte de la SRS de un proyecto software.

- RESTRICCIONES

  Solo se podrá modificar la descripción de un atributo que haga parte de la definición de un requerimiento creado previamente; El Id de un requerimiento no podrá ser cambiado.

- PROCESO
  - El usuario selecciona la opción Modificar Requerimientos.
  - El sistema presenta la información de los requerimientos de un proyecto para que puedan ser seleccionados y modificados.
  - El sistema debe validar que no se encuentren vacíos los campos que se consideren obligatorios, para la definición de un requerimiento.
  - Luego de ingresar la información, el usuario hace clic en la opción Guardar.
  - Si la información ha sido ingresada correctamente, se actualiza la información en la base de datos.

- ENTRADAS DEL PROCESO

  Todas las entradas del proceso están dadas por las solicitadas en el formulario desplegado por el sistema, que tienen que ver con la actualización de un requerimiento específico.

- SALIDAS DEL PROCESO

  La salida que se produce está directamente ligada con la actualización de los datos en la tabla correspondiente en la base de datos.

  En caso que los datos no sean ingresados correctamente, se emitirán los respectivos mensajes de error.

4.4.4. Generación de documentación de Requerimientos

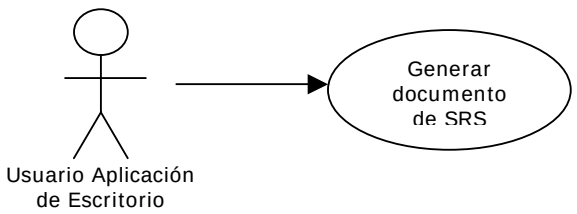


Figura 7. Caso de Uso Generación de Documentación de Requerimientos

4.4.4.1. CASO DE USO Generar Documento de SRS

- DESCRIPCIÓN
  En este caso de uso se permite la generación del documento de la especificación de requerimientos de software SRS de un proyecto determinado, en formato RTF.

- RESTRICCIONES
  Debe existir un proyecto registrado par poder generar la documentación.

- PROCESO
  - El usuario selecciona la opción Generar informe de SRS.
  - El sistema valida el proyecto activo en el sistema.
  - El sistema genera el documento de la SRS del proyecto.

- ENTRADAS DEL PROCESO
  El usuario selecciona la opción para generar el documento de SRS.

- SALIDAS DEL PROCESO
  Se debe obtener un archivo donde se almacene la SRS generada.

4.4.5. Sincronización de datos entre las dos aplicaciones

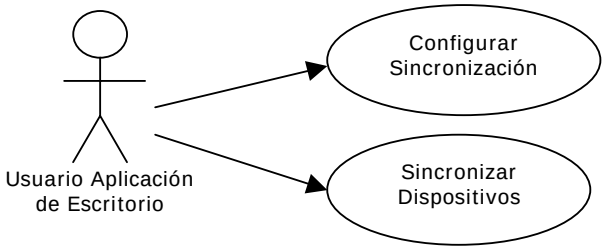


Figura 8. Caso de Uso Sincronización de Datos

4.4.5.1. CASO DE USO Configurar Sincronización

- DESCRIPCIÓN
  En este caso de uso se permite configurar e instalar la aplicación móvil y las tablas de datos de la aplicación.

- RESTRICCIONES

  Debe existir los archivos correspondientes para el proceso de instalación y sincronización. Además la aplicación que administra la sincronización del dispositivo debe estar instalada en el computador de escritorio y en ejecución, con el registro del dispositivo que se va a sincronizar.

- PROCESO
  - El usuario selecciona la opción para instalar la aplicación móvil desde la aplicación de escritorio.
  - Luego, selecciona el dispositivo correspondiente al que se le va a instalar la aplicación.
  - El usuario selecciona la opción Aceptar.
  - El sistema transfiere al PDA lo que el usuario le ha ordenado instalar.

- ENTRADAS DEL PROCESO

  El usuario selecciona el dispositivo para realizar la instalación.

- SALIDAS DEL PROCESO

  El dispositivo móvil se encuentra actualizado con la nueva instalación.

4.4.5.2. CASO DE USO Sincronizar Dispositivos

- DESCRIPCIÓN

  Permite realizar la sincronización para la actualización de datos entre la aplicación móvil y la de escritorio.

- RESTRICCIONES

  Debe existir los archivos correspondientes para el proceso de sincronización. Además la aplicación Hotsync Manager debe estar instalada en el computador de escritorio y en ejecución, con el registro del dispositivo que se va a sincronizar.

- PROCESO
  - El usuario selecciona la opción para sincronizar la aplicación desde la aplicación de escritorio.
  - El sistema ejecuta la sincronización a través del conduit
  - El sistema realiza el intercambio y actualización de datos correspondiente.

- ENTRADAS DEL PROCESO

  El usuario selecciona la opción para realizar la sincronización.

- SALIDAS DEL PROCESO

  La salida que se produce está directamente ligada con la actualización de los datos en la tabla correspondiente en la base de datos.

4.4.6. Mantenimiento del Sistema

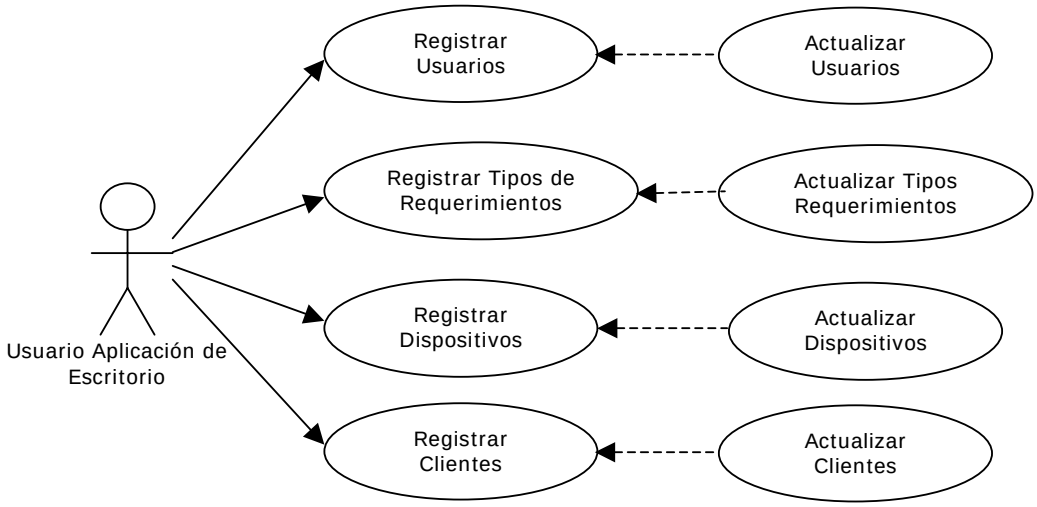


Figura 9. Caso de Uso Mantenimiento del Sistema

4.4.6.1. CASO DE USO Registrar Usuarios

- DESCRIPCIÓN

  En este caso de uso se permite registrar, modificar o eliminar los usuarios que intervienen en la SRS.

- RESTRICCIONES

  El único usuario autorizado para realizar las tareas de registro y actualización mencionadas es el administrador del sistema. No se podrá eliminar un usuario que tenga registros relacionados.

- PROCESO
  - El usuario selecciona la opción Usuarios.
  - El usuario ingresa los datos correspondientes para registrar un nuevo usuario (o para actualizar).
  - Luego de validados los datos por el sistema, el usuario guarda la información.
  - Los datos son insertados (o actualizados) en las tablas correspondiente en la base de datos.

- ENTRADAS DEL PROCESO

  El usuario ingresa los campos correspondientes dentro de un formulario para definir un nuevo usuario (o para modificarlo).

- SALIDAS DEL PROCESO

  Los datos son insertados en las tablas correspondientes en la base de datos, o son actualizados, si el usuario ya estaba registrado previamente.

  Si los datos son ingresados de manera errónea, se deben emitir los correspondientes mensajes de error.

NOTA: El proceso para registro, modificación y eliminación de clientes mantiene una estructura similar, por lo que se emite la descripción de éste caso de uso.

4.4.6.2. CASO DE USO Registrar tipos de requerimientos

- DESCRIPCIÓN

  En este caso de uso se permite registrar, o modificar las tablas de soporte de los diferentes tipos de requerimientos con sus atributos.

- RESTRICCIONES

  Los usuarios autorizados para realizar las tareas de registro y actualización mencionadas son el administrador del sistema y el ingeniero de requisitos/analista de sistemas (que puede ser el mismo administrador del sistema).

- PROCESO
  - El usuario selecciona la opción del menú correspondiente a requerimientos.
  - El usuario ingresa los datos correspondientes para registrar (o para actualizar) un nuevo tipo de requerimiento y agrega sus atributos.
  - Luego de validados los datos por el sistema, el usuario guarda la información.
  - Los datos son insertados (o actualizados) en las tablas correspondiente en la base de datos.

- ENTRADAS DEL PROCESO

  El usuario ingresa los campos correspondientes dentro de un formulario para definir un nuevo tipo de requerimiento, junto con sus atributos (o para modificarlos).

- SALIDAS DEL PROCESO

  Los datos son insertados en las tablas correspondientes en la base de datos, o son actualizados, si el tipo de requerimiento ya estaba definido previamente.

  Si los datos son ingresados de manera errónea, se deben emitir los correspondientes mensajes de error.

4.4.6.3. CASO DE USO Registrar Dispositivos

- DESCRIPCIÓN

  Permite registrar, modificar o eliminar los dispositivos que han de ser sincronizados con la aplicación de escritorio.

- RESTRICCIONES

  El usuario autorizado para realizar las tareas de registro y actualización mencionadas es el administrador del sistema.

- PROCESO
  - El usuario selecciona la opción Dispositivos.
  - El usuario ingresa los datos correspondientes para registrar un nuevo dispositivo (o para actualizar).
  - Luego de validados los datos por el sistema, el usuario guarda la información.
  - Los datos son insertados (o actualizados) en las tablas correspondiente en la base de datos.

- ENTRADAS DEL PROCESO

  El usuario ingresa los campos correspondientes dentro de un formulario para definir un nuevo dispositivo (o para modificarlo).

- SALIDAS DEL PROCESO

  Los datos son insertados en las tablas correspondientes en la base de datos, o son actualizados, si el dispositivo ya estaba registrado previamente.

  Si los datos son ingresados de manera errónea, se deben emitir los correspondientes mensajes de error.

5. CONCLUSIONES

Pese a que los problemas en la industria del software persisten, no se le ha dado el lugar correspondiente a las actividades que involucra la Ingeniería de Requerimientos ni a los procesos de desarrollo de software que propone la Ingeniería del Software. No es suficiente que esto procesos, actividades, metodologías y estándares existan, están ahí para que esa industria que se encarga de proveer soluciones software, sea responsable de su propia preparación y pueda enfrentar con altura la demanda del mercado, que exige calidad.

Teniendo en cuenta la gran importancia que tiene la especificación de requerimientos para el desarrollo de un proyecto software, este proyecto de grado tuvo como objetivo central el diseño e implementación de un prototipo software para la captura, especificación y documentación de requerimientos de software utilizando un estándar reconocido en el área de la ingeniería del software. Dentro de los estándares existentes, se seleccionó el estándar IEEE 830, que aunque no es recomendable para proyectos que utilicen metodologías DRA, por el trabajo detallado y responsable que implica, es un estándar que se puede adaptar fácilmente a las necesidades de una organización. El producto obtenido permite entonces administrar los requerimientos de un proyecto software de manera flexible y ser administrado para hacerlo más adaptable a los procesos que maneje la organización que lo utilice.

Este proyecto no estuvo enfocado a una entidad específica, pero quiso fomentar el interés por el trabajo en la línea de investigación de Ingeniería del Software en la Escuela de Ingeniería de Sistemas e Informática. Es por esto, que éste trabajo de grado va dirigido a cualquier organización cuyo objeto sea la elaboración y el desarrollo de soluciones software, que pretendan aplicar metodologías para el mejoramiento de las fases tempranas del desarrollo del software, en especial que se inclinen por una mejor administración de los requerimientos de sus proyectos software. En nuestro medio se hace necesario tomar conciencia que la elaboración de proyectos software no puede seguirse haciendo de manera artesanal. Se necesita personal capacitado para dirigir proyectos informáticos y que estén preparados entre otras cosas, en las áreas de ingeniería del software y afines, de modo que puedan dar un mejor cauce a los proyectos. La ingeniería de requerimientos no es un proceso aislado, requiere del trabajo conjunto de ingenieros, clientes, usuarios, administradores, equipo de desarrollo, que debe tener un nivel de experiencia mínimo y el dominio del área del problema, para conseguir los resultados esperados por las partes involucradas.

Se puede poner las tecnologías de la información al servicio de las tecnologías mismas, donde el desarrollo de soluciones informáticas se sirve de recursos tecnológicos para su propio beneficio.

6. RECOMENDACIONES

El trabajo realizado en este proyecto de grado es susceptible de mejoras y por lo tanto se presentan algunas recomendaciones para trabajos futuros en ésta área de la ingeniería del software:

Desarrollar una herramienta para la administración de requerimientos de software que proporcione además de la definición textual de requerimientos, la definición gráfica utilizando casos de uso.

Facilitar la incorporación y adaptación con otras herramientas utilizadas en el proceso de desarrollo de software para lograr un trabajo conjunto y productivo.

Implementar soluciones utilizando estándares o metodologías propuestas en el área de la Ingeniería del Software, o realizar adaptaciones que ayuden a mejorar los procesos en las fases tempranas del desarrollo del software.

No está de más incursionar creativamente con nuevas metodologías que aporten efectivamente al proceso de desarrollo de software en cualquiera de sus etapas.

7. REFERENCIAS BIBLIOGRÁFICAS

🕮 PRESSMAN, Roger. Ingeniería del Software – Un enfoque práctico, Cuarta Edición. McGraw Hill, España, 1998. (Quinta Edición - 2002)

🕮 SOMMERVILLE, Ian. Ingeniería del Software, Sexta Edición. Pearson Education, México, 2002.

🕮 MOORE, James W. Software Engineering Standards, A User's Road Map. IEEE Computer Society, Unites States, 2000.

🕮 McCONNELL, Steve. Desarrollo y gestión de proyectos informáticos. McGraw Hill, España, 1997.

🕮 Software Engineering Standards Committee of the IEEE Computer Society. IEEE Std. 830-1998 IEEE Recommended Practice for Software Requirements Specifications. The Institute of Electrical and Electronics Engineers, Inc., United States, 1998.

🕮 Software Engineering Standards Committee of the IEEE Computer Society. IEEE Std. 1074-1997 IEEE Standard for Developing Software Life Cycle Processes. The Institute of Electrical and Electronics Engineers, Inc., United States, 1998.

🕮 ZAPATA J., Carlos M., ARANGO ISAZA, J. Fernando. Alineación entre Metas Organizacionales y Elicitación de Requisitos del Software. Dyna, Año 71, Nro. 143, pags. 101-110. Medellín, Noviembre de 2004.

🕮 ADIBOWO, Sasmito. RAMBUTAN, Requirements management tool for busy systems analysts. Faculty of Computer Science, University of Indonesia, 2003.

🕮 IRqA® 2.0 El Control de sus Especificaciones. © TCP Sistemas e Ingeniería, Abril de 2002.

🕮 PÉREZ ROJAS, Adriana Rocío. Plan de Proyecto de Grado PASCAR. Universidad Industrial de Santander, Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería de Sistemas, Noviembre de 2005.

8. ANEXOS

8.1. ESTANDAR IEEE 830

El estándar IEEE 830-1998 (IEEE Recommended Practice for Software Requirements Specifications) se presenta a continuación en las siguientes páginas.

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

ISBN 0-7381-0332-2

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.

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

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.[1]

IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.[2]

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.[3]

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.[4]

IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions.[5]

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.[6]

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

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

IEEE Std 1233, 1998 Edition, IEEE Guide for Developing System Requirements Specifications.[8]

---

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

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

[3]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.

[4]See Footnote 3.

[5]See Footnote 3.

[6]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.

[7]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.

[8]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 nontechnical 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 × 60% of the time; and shall be produced within 30 s of event × 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.

Table of Contents

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 that “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 a) Support the life cycle data objectives of Annex H of IEEE/EIA 12207.0-1996	—	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.

8.2. DICCIONARIO DE DATOS

8.2.1. Diccionario de Datos Aplicación de Escritorio

- **Nombre Tabla:** Tbl_CLIENTES
- **Descripción:** Almacena los datos identificadores de los clientes.

Tbl_CLIENTES			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Id_Cliente	VARCHAR	10	Identificador único para el cliente
Nombre	VARCHAR	100	Nombre completo del cliente o representante de la empresa
Empresa	VARCHAR	50	Nombre de la Empresa del cliente
Direccion	VARCHAR	100	Dirección principal de la empresa cliente
Telefono	VARCHAR	10	Teléfono del cliente
Celular	VARCHAR	15	Teléfono móvil del cliente
Ciudad	VARCHAR	50	Ciudad de residencia del cliente
Departamento	VARCHAR	50	Departamento de residencia del cliente
Email	VARCHAR	20	Correo electrónico del cliente
Fecha_Crea	VARCHAR	10	Fecha de creación del cliente
Fecha_Modif	VARCHAR	10	Fecha de modificación del cliente
Hora_Modif	VARCHAR	10	Hora de modificación del cliente
Usuario_Modif	VARCHAR	10	Código del Usuario que modificó el cliente

Tabla 2. Tabla Clientes Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_TIPOS_DESCRIP_PROY
- **Descripción:** Almacena los tipos de descripción que se utilizan para detallar un proyecto.

Tbl_TIPOS_DESCRIP_PROY			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Tipo	VARCHAR	10	Identifica un tipo de descripción de un proyecto
Nombre	VARCHAR	50	Nombre del tipo de descripción
Descripcion	VARCHAR	100	Descripción del tipo de descripción de un proyecto

Tabla 3. Tabla Tipos de Atributos Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_ATRIBUTOS_TIPOS_DESCRIP_PROY
- **Descripción:** Almacena los atributos correspondientes a un tipo de descripción de proyectos.

Tbl_ATRIBUTOS_TIPOS_DESCRIP_PROY			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Atributo	VARCHAR	10	Identifica el atributo que pertenece a un tipo de descripción de un proyecto
Nombre	VARCHAR	50	Nombre del atributo

Descripcion	VARCHAR	100	Descripción del atributo
Codigo_Tipo	VARCHAR	10	Identifica a que tipo de descripción pertenece el atributo

Tabla 4. Tabla Atributos de Tipos de Atributos Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_PROYECTOS
- **Descripción:** Almacena los datos identificadores de cada proyecto.

Tbl_PROYECTOS			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Proyecto	VARCHAR	6	Código único alfa-numérico que representa la identificación de un proyecto.
Nombre	VARCHAR	100	Nombre del proyecto
Fecha_Creacion	VARCHAR	10	Fecha de creación del proyecto en formato aaaa/mm/dd
Hora_Creacion	VARCHAR	10	Hora de creación del proyecto en formato hh:mm:ss
Id_Cliente	VARCHAR	10	Identifica el cliente del proyecto software
Fecha_Modif	VARCHAR	10	Fecha de modificación de los datos del proyecto
Hora_Modif	VARCHAR	10	Hora de modificación de los datos del proyecto
Usuario_Modif	VARCHAR	10	Código del Usuario que modificó los datos del proyecto

Tabla 5. Tabla Proyectos Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_PROYECTOS_ATRIBUTOS
- **Descripción:** Almacena la descripción de un atributo perteneciente a un proyecto.

Tbl_PROYECTOS_ATRIBUTOS			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Id_Atributo	VARCHAR	12	Identificador único del atributo dentro de un proyecto determinado
Codigo_Proyecto	VARCHAR	6	Código único alfa-numérico que representa la identificación de un proyecto.
Codigo_Atributo	VARCHAR	10	Identifica el atributo que pertenece a un tipo de descripción de un proyecto
Descripción	LONGCHAR		Descripción del proyecto en función del atributo seleccionado
Fecha_Crea	VARCHAR	10	Fecha de creación del atributo del proyecto
Hora_Crea	VARCHAR	10	Hora de creación del atributo del proyecto
Fecha_Modif	VARCHAR	10	Fecha de modificación del atributo del proyecto
Hora_Modif	VARCHAR	10	Hora de modificación del atributo del proyecto
Usuario_Modif	VARCHAR	10	Usuario que modificó el atributo del proyecto

Tabla 6. Tabla Atributos de Proyectos Aplicación de escritorio Pascar

- Tbl_TIPOS_REQ
- **Descripción:** Almacena los tipos de requerimientos que se utilizan para detallar un proyecto.

Tbl_TIPOS_REQ			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Tipo	VARCHAR	10	Identifica un tipo de requerimiento
Nombre	VARCHAR	50	Nombre del tipo de requerimiento
Descripcion	VARCHAR	100	Descripción del tipo de requerimiento

Tabla 7. Tabla Tipos de Requerimientos Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_ATRIBUTOS_TIPOS_REQ
- **Descripción:** Almacena los atributos correspondientes a un tipo de requerimientos.

Tbl_ATRIBUTOS_TIPOS_REQ			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Atributo	VARCHAR	10	Identifica el atributo de un tipo de requerimiento determinado
Nombre	VARCHAR	50	Nombre del atributo de un tipo de requerimiento
Descripcion	VARCHAR	100	Describe el atributo de un tipo de requerimiento
Codigo_Tipo	VARCHAR	10	Identifica el tipo de requerimiento al que pertenece el atributo

Tabla 8. Tabla Atributos de Tipos de Requerimientos Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_REQUERIMIENTOS
- **Descripción:** Almacena los requerimientos de cada proyecto.

Tbl_REQUERIMIENTOS			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Id_Requerimiento	VARCHAR	12	Identifica un requerimiento determinado
Nombre	VARCHAR	50	Nombre del requerimiento
Descripción	LONGCHAR		Describe el requerimiento
Codigo_Atributo	VARCHAR	10	Identifica el atributo de un tipo de requerimiento determinado
Codigo_Proyecto	VARCHAR	6	Código único alfa-numérico que representa la identificación de un proyecto.
Fecha_Crea	VARCHAR	10	Fecha de creación del requerimiento en formato aaaa/mm/dd
Hora_Crea	VARCHAR	10	Hora de creación del requerimiento en formato hh:mm:ss
Fecha_Modif	VARCHAR	10	Fecha de modificación del requerimiento en formato aaaa/mm/dd
Hora_Modif	VARCHAR	10	Hora de modificación del requerimiento en formato hh:mm:ss
Usuario_Modif	VARCHAR	10	Usuario que modificó el requerimiento

Tabla 9. Tabla Requerimientos de Proyectos Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_TIPO_USUARIO
- **Descripción:** Almacena los tipos de usuarios que intervienen en el documento de especificación de requisitos de software del proyecto.

Tbl_TIPO_USUARIO			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Tipo	VARCHAR	10	Identifica un tipo de usuario
Nombre	VARCHAR	50	Nombre del tipo de usuario
Descripción	VARCHAR	100	Descripción del tipo de usuario

Tabla 10. Tabla Tipos de Usuarios Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_USUARIOS
- **Descripción:** Almacena los datos identificadores de los usuarios.

Tbl_USUARIOS			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Usuario	VARCHAR	10	Identifica a un usuario determinado. Puede ser su NIT o cédula de identificación.
Nombres	VARCHAR	50	Nombre(s) del usuario
Apellidos	VARCHAR	50	Apellido(s) del usuario
Login	VARCHAR	10	Login de autenticación del usuario
Password	VARCHAR	10	Contraseña de autenticación del usuario
Codigo_Tipo	VARCHAR	10	Identifica el tipo de usuario

Tabla 11. Tabla Usuarios Aplicación de escritorio Pascar

- Tbl_DISPOSITIVOS
- **Descripción:** Almacena la identificación de los dispositivos disponibles.

Tbl_DISPOSITIVOS			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Nombre	VARCHAR	50	Nombre del dispositivo
Nota	LONGCHAR		Anotación/Descripción del dispositivo
Codigo_Usuario	VARCHAR	10	Código del usuario responsable del dispositivo

Tabla 12. Tabla Dispositivos Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_RECOMENDACIONES
- **Descripción:** Almacena las recomendaciones realizadas dentro de un proyecto por los usuarios.

Tbl_RECOMENDACIONES			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Recomend	VARCHAR	10	Código que identifica la recomendación realizada al proyecto por un usuario
Nombre_Recomend	LONGCHAR		Nombre descriptivo de la recomendación realizada

Descripcion	LONGCHAR		Descripción detallada de la recomendación realizada al proyecto
Fecha_Crea	VARCHAR	10	Fecha en la que se hace la recomendación
Hora_Crea	VARCHAR	10	Hora en la que se hace la recomendación
Codigo_Usuario	VARCHAR	10	Codigo que identifica al usuario que hace la recomendación
Codigo_Proyecto	VARCHAR	10	Codigo del proyecto al que pertenece la recomendación

Tabla 13. Tabla Recomendaciones Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_PROYECTOS_USUARIOS
- **Descripción:** Identifica la relación existente entre un proyecto y sus usuarios.

Tbl_PROYECTOS_USUARIOS			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Relacion	VARCHAR	10	Código único de identificación de la relación proyecto-usuario
Codigo_Proyecto	VARCHAR	10	Identifica un proyecto determinado
Codigo_Usuario	VARCHAR	10	Identifica a un usuario de un proyecto

Tabla 14. Tabla Proyectos-Usuarios Aplicación de escritorio Pascar

- **Nombre Tabla:** Tbl_AUDITORIA
- **Descripción:** Es la bitácora de la aplicación, donde se almacenan los cambios que un usuario ha realizado dentro de de la SRS de un proyecto.

Tbl_AUDITORIA			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Usuario	VARCHAR	10	Código para identificar el usuario que ha realizado los cambios
Codigo_Evento	VARCHAR	10	Codigo del evento que se genera al realizar un cambio
Fecha	VARCHAR	10	Fecha en la cual se realiza un cambio dentro de la especificación de req. del proyecto
Hora	VARCHAR	10	Hora en la cual se realiza el cambio
Codigo_Proyecto	VARCHAR	6	Codigo del proyecto al cual se realiza el cambio en la especificación de req. del software

Tabla 15. Tabla Auditoria Aplicación de escritorio Pascar

8.2.2. Diccionario de Datos Aplicación Móvil

- **Nombre Tabla:** M_USUARIOS
- **Descripción:** Almacena los datos identificadores de los usuarios.

M_USUARIOS			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Codigo_Usu	VARCHAR	10	Identifica a un usuario determinado
Nombre_usu	VARCHAR	100	Nombre completo del usuario
Password	VARCHAR	10	Contraseña de autenticación del usuario
Nombre_Dis	VARCHAR	50	Nombre único del dispositivo
Nota	LONGCHAR		Anotación/Descripción del dispositivo

Tabla 16. Tabla Usuarios Aplicación móvil Pascar

- **Nombre Tabla:** M_CLIENTES
- **Descripción:** Almacena los datos identificadores de los clientes.

M_CLIENTES			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Id_Cliente	VARCHAR	10	Identificador único para el cliente
Nombre	VARCHAR	100	Nombre completo del cliente o representante de la empresa
Empresa	VARCHAR	50	Nombre de la Empresa del cliente
Direccion	VARCHAR	100	Dirección principal de la empresa cliente
Telefono	VARCHAR	10	Teléfono del cliente
Celular	VARCHAR	15	Teléfono móvil del cliente
Ciudad	VARCHAR	50	Ciudad de residencia del cliente
Departam	VARCHAR	50	Departamento de residencia del cliente
Email	VARCHAR	20	Correo electrónico del cliente

Tabla 17. Tabla Clientes Aplicación móvil Pascar

- **Nombre Tabla:** M_TIPOS_DESCRIP_PROY
- **Descripción:** Almacena los tipos de descripción que se utilizan para detallar un proyecto.

M_TIPOS_DESCRIP_PROY			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Cod_Tipo	VARCHAR	10	Identifica un tipo de descripción de un proyecto
Nombre	VARCHAR	50	Nombre del tipo de descripción

Tabla 18. Tabla Tipos de Atributos Aplicación móvil Pascar

- **Nombre Tabla:** M_ATRIBUTOS_TIPOS_DE (M_ATRIBUTOS_TIPOS_DESCRIP_PROY)
- **Descripción:** Almacena los atributos correspondientes a un tipo de descripción de proyectos.

M_ATRIBUTOS_TIPOS_DE			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Cod_Atrib	VARCHAR	10	Identifica el atributo que pertenece a un tipo de descripción de un proyecto
Nombre	VARCHAR	50	Nombre del atributo
Cod_Tipo	VARCHAR	10	Identifica a que tipo de descripción pertenece el atributo

Tabla 19. Tabla Atributos de Tipos de Atributos Aplicación móvil Pascar

- **Nombre Tabla:** M_PROY_ATRIBUTOS (M_PROYECTOS_ATRIBUTOS)
- **Descripción:** Almacena la descripción de un atributo perteneciente a un proyecto.

M_PROY_ATRIBUTOS			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Id_Atrib	VARCHAR	12	Identificador único del atributo dentro de un proyecto determinado
Cod_Proy	VARCHAR	6	Código único alfa-numérico que representa la identificación de un proyecto
Cod_Atrib	VARCHAR	10	Identifica el atributo que pertenece a un tipo de descripción de un proyecto
Nom_Atrib	VARCHAR	50	Nombre del atributo
Descrip	LONGCHAR		Descripción del proyecto en función del atributo seleccionado
Fecha_Mod	VARCHAR	10	Fecha de modificación del atributo del proyecto
Hora_Mod	VARCHAR	10	Hora de modificación del atributo del proyecto
Usu_Mod	VARCHAR	10	Usuario que modificó el atributo del proyecto

Tabla 20. Tabla Atributos de Proyectos Aplicación móvil Pascar

- **Nombre Tabla:** M_PROYECTOS
- **Descripción:** Almacena los datos identificadores de cada proyecto.

M_PROYECTOS			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Cod_Proy	VARCHAR	6	Código único alfa-numérico que representa la identificación de un proyecto.
Nombre	VARCHAR	100	Nombre del proyecto
Id_Cliente	VARCHAR	10	Identifica el cliente del proyecto software
Nombre_Cli	VARCHAR	100	Nombre del cliente del proyecto

Tabla 21. Tabla Proyectos Aplicación móvil Pascar

- **Nombre Tabla:** M_TIPOS_REQ
- **Descripción:** Almacena los tipos de requerimientos que se utilizan para detallar un proyecto.

M_TIPOS_REQ			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Cod_Tipo	VARCHAR	10	Identifica un tipo de requerimiento
Nombre	VARCHAR	50	Nombre del tipo de requerimiento

Tabla 22. Tabla Tipos de Requerimientos Aplicación móvil Pascar

- **Nombre Tabla:** M_ATRIBUTOS_TIPOS_RE (M_ATRIBUTOS_TIPOS_REQ)
- **Descripción:** Almacena los atributos correspondientes a un tipo de requerimientos.

M_ATRIBUTOS_TIPOS_RE			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Cod_Atrib	VARCHAR	10	Identifica el atributo de un tipo de requerimiento determinado
Nombre	VARCHAR	50	Nombre del atributo de un tipo de requerimiento
Cod_Tipo	VARCHAR	10	Identifica el tipo de requerimiento al que pertenece el atributo

Tabla 23. Tabla Atributos de Requerimientos Aplicación móvil Pascar

- **Nombre Tabla:** M_REQUERIMIENTOS
- **Descripción:** Almacena los requerimientos de cada proyecto.

M_REQUERIMIENTOS			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Id_Requer	VARCHAR	12	Identifica un requerimiento determinado
Nombre	VARCHAR	50	Nombre del requerimiento
Descrip	LONGCHAR		Describe el requerimiento
Cod_Atrib	VARCHAR	10	Identifica el atributo de un tipo de requerimiento determinado
Nom_Atrib	VARCHAR	50	Nombre del atributo de un tipo de requerimiento
Cod_Proy	VARCHAR	6	Identifica el proyecto al que pertenece el requerimiento
Fecha_Mod	VARCHAR	10	Fecha de modificación del requerimiento en formato aaaa/mm/dd
Hora_Mod	VARCHAR	10	Hora de modificación del requerimiento en formato hh:mm:ss
Usu_Mod	VARCHAR	10	Usuario que modificó el requerimiento

Tabla 24. Tabla Requerimientos de Proyectos Aplicación móvil Pascar

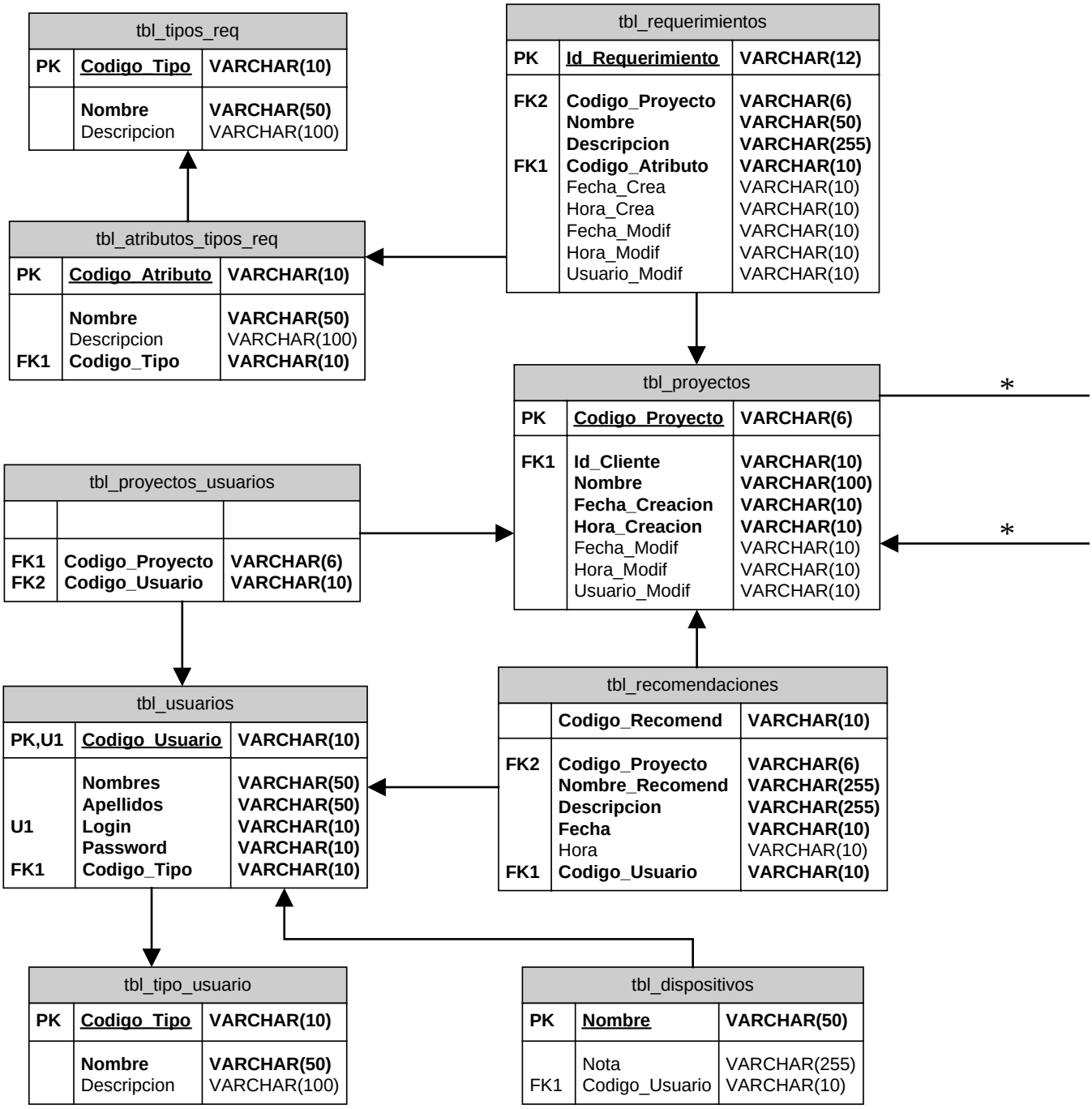
- **Nombre Tabla:** M_AUDITORIA
- **Descripción:** Es la bitácora de la aplicación, donde se almacenan los cambios que un usuario ha realizado dentro de de la SRS de un proyecto.

M_AUDITORIA			
CAMPO	**TIPO**	**LONG**	**DESCRIPCIÓN**
Cod_Usu	VARCHAR	10	Código para identificar el usuario que ha realizado los cambios
Cod_Evento	VARCHAR	10	Codigo del evento que se genera al realizar un cambio
Fecha	VARCHAR	10	Fecha en la cual se realiza un cambio dentro de la especificación de req. del proyecto
Hora	VARCHAR	10	Hora en la cual se realiza el cambio
Cod_Proy	VARCHAR	6	Codigo del proyecto al cual se realiza el cambio en la especificación de req. del software

Tabla 25. Tabla Auditoria Aplicación móvil Pascar

8.3. MODELO DE DATOS

8.3.1. Modelo de Datos Aplicación de Escritorio



tbl_auditoria		
	Codigo_Usuario	**VARCHAR(10)**
	Codigo_Evento	**VARCHAR(10)**
	Codigo_Proyecto	**VARCHAR(6)**
	Fecha	**VARCHAR(10)**
	Hora	**VARCHAR(10)**

tbl_clientes		
PK	**Id_Cliente**	**VARCHAR(10)**
	Nombre	VARCHAR(100)
	Empresa	VARCHAR(50)
	Direccion	VARCHAR(100)
	Telefono	VARCHAR(10)
	Celular	VARCHAR(15)
	Ciudad	VARCHAR(50)
	Departamento	VARCHAR(50)
	Email	VARCHAR(20)
	Fecha_Crea	VARCHAR(10)
	Fecha_Modif	VARCHAR(10)
	Hora_Modif	VARCHAR(10)
	Usuario_Modif	VARCHAR(10)

*

*

tbl_proyectos_atributos		
PK	**Id_Atributo**	**VARCHAR(12)**
FK2	**Codigo_Proyecto**	**VARCHAR(6)**
FK1	**Codigo_Atributo**	**VARCHAR(10)**
	Descripcion	**VARCHAR(255)**
	Fecha_Crea	**VARCHAR(10)**
	Hora_Crea	**VARCHAR(10)**
	Fecha_Modif	**VARCHAR(10)**
	Hora_Modif	**VARCHAR(10)**
	Usuario_Modif	**VARCHAR(10)**

tbl_atributos_tipos_descrip_proy		
PK	**Codigo_Atributo**	**VARCHAR(10)**
	Nombre	**VARCHAR(50)**
	Descripcion	VARCHAR(100)
FK1	**Codigo_Tipo**	**VARCHAR(10)**

tbl_tipos_descrip_proy		
PK	**Codigo_Tipo**	**VARCHAR(10)**
	Nombre	VARCHAR(50)
	Descripcion	VARCHAR(100)

8.3.2. Modelo de Datos Aplicación Móvil

m_usuarios		
PK,U1	**Codigo_Usu**	**VARCHAR(10)**
	Nombre_Usu	**VARCHAR(100)**
	Password	**VARCHAR(10)**
	Nombre_Dis	**VARCHAR(50)**
	Nota	VARCHAR(255)

m_tipos_req		
PK	**Cod_Tipo**	**VARCHAR(10)**
	Nombre	**VARCHAR(50)**

m_atributos_tipos_re		
PK	**Cod_Atrib**	**VARCHAR(10)**
	Nombre	**VARCHAR(50)**
	Cod_Tipo	**VARCHAR(10)**

m_proy_atributos		
PK	**Id_Atrib**	**VARCHAR(12)**
	Cod_Proy	**VARCHAR(6)**
	Cod_Tipo	**VARCHAR(10)**
	Cod_Atrib	**VARCHAR(10)**
	Nom_Atrib	**VARCHAR(50)**
	Descrip	**VARCHAR(255)**
	Fecha_Mod	**VARCHAR(10)**
	Hora_Mod	**VARCHAR(10)**
	Usu_Mod	**VARCHAR(10)**

m_auditoria		
	Cod_Usua	**VARCHAR(10)**
	Cod_Evento	**VARCHAR(10)**
	Cod_Proy	**VARCHAR(6)**
	Fecha	**VARCHAR(10)**
	Hora	**VARCHAR(10)**

m_clientes		
PK	**Id_Cliente**	**VARCHAR(10)**
	Nombre	VARCHAR(100)
	Empresa	VARCHAR(50)
	Direccion	VARCHAR(100)
	Telefono	VARCHAR(10)
	Celular	VARCHAR(15)
	Ciudad	VARCHAR(50)
	Departamento	VARCHAR(50)
	Email	VARCHAR(20)

m_requerimientos		
PK	**Id_Requer**	**VARCHAR(12)**
	Cod_Proy	**VARCHAR(6)**
	Nombre	**VARCHAR(50)**
	Descrip	**VARCHAR(255)**
	Cod_Atrib	**VARCHAR(10)**
	Nom_Atrib	VARCHAR(50)
	Fecha_Mod	VARCHAR(10)
	Hora_Mod	VARCHAR(10)
	Usu_Mod	VARCHAR(10)

m_proyectos		
PK	**Cod_Proy**	**VARCHAR(6)**
	Nombre	**VARCHAR(100)**
	Id_Cliente	**VARCHAR(10)**
	Nombre_Cli	VARCHAR(100)

m_atributos_tipos_de		
PK	**Cod_Atrib**	**VARCHAR(10)**
	Nombre	**VARCHAR(50)**
	Cod_Tipo	**VARCHAR(10)**

m_tipos_descrip_proy		
PK	**Cod_Tipo**	**VARCHAR(10)**
	Nombre	VARCHAR(50)

8.4. MANUALES DE USUARIO

8.4.1. Manual de Usuario Aplicación de Escritorio

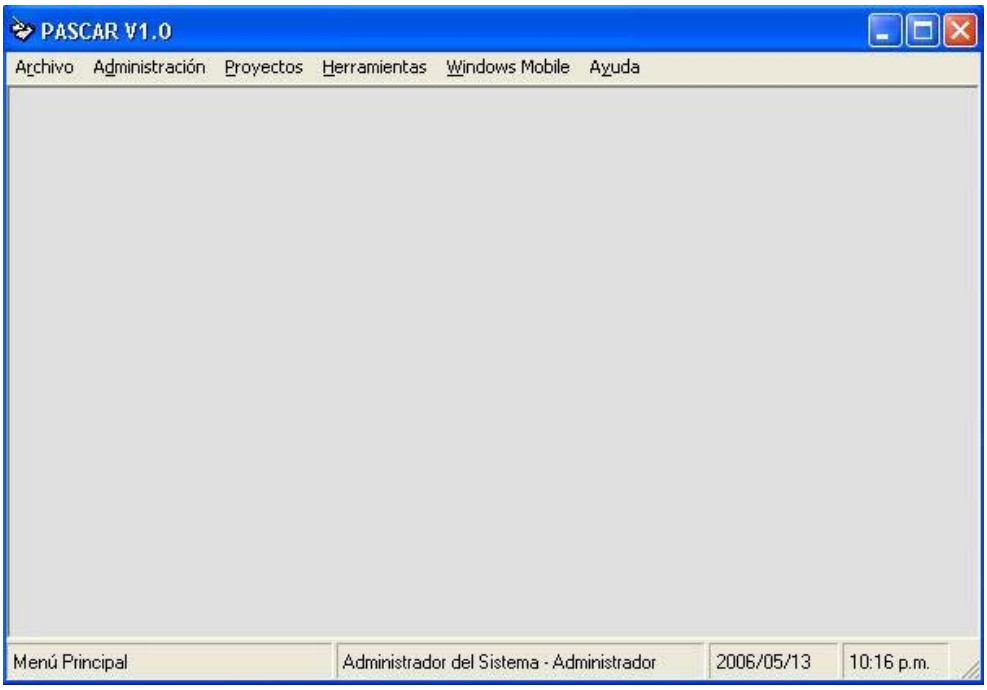
8.4.1.1 Inicio de la Aplicación

Cuando se inicia la aplicación PASCAR, aparece la ventana de autenticación de usuarios, donde se deben ingresar el nombre de usuario (Login) y la contraseña (password) del usuario que va a ingresar a la aplicación:

Luego se selecciona la opción Aceptar para validar si el usuario está o no registrado en el sistema. En el primer caso se dará acceso a la interfaz principal de la aplicación, y en el segundo caso se emitirá un mensaje de error y se restringe el acceso.

8.4.1.2 Interfaz Principal

La interfaz principal cuenta con 6 opciones de menú. Cada menú cuenta con diferentes opciones que son habilitadas de acuerdo al perfil de usuario definido, ya sea administrador (con todas las opciones de la aplicación disponibles), ingeniero de requerimientos (con restricción a las opciones administrativas de mantenimiento) o ingeniero de desarrollo (el cual solo puede acceder al modulo de recomendaciones). Las opciones de menú son las siguientes:

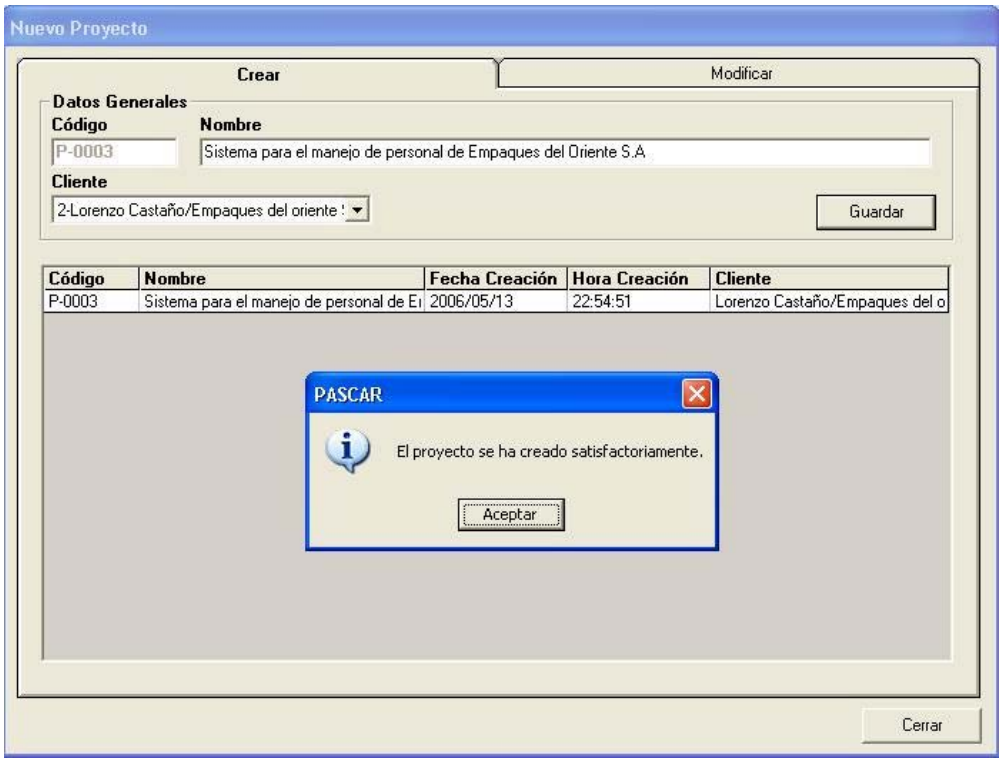


- Menú Archivo
  - Nuevo Proyecto
  - Abrir Proyecto
  - Cerrar Sesión
  - Salir
- Menú Administración
  - Tipos de Atributos
  - Atributos
  - Tipos de Requerimientos
  - Atributos de Tipos de Requerimientos
  - Usuarios
  - Clientes
  - Dispositivos
- Menú Proyectos
  - Atributos del Proyecto
  - Requerimientos del Proyecto
- Menú Herramientas
  - Generar Informe SRS
  - Recomendaciones
- Menú Windows Mobile
  - Sincronizador de Dispositivos
- Menú Ayuda
  - Manual Aplicación Móvil

- Manual Aplicación de Escritorio
- Acerca de…

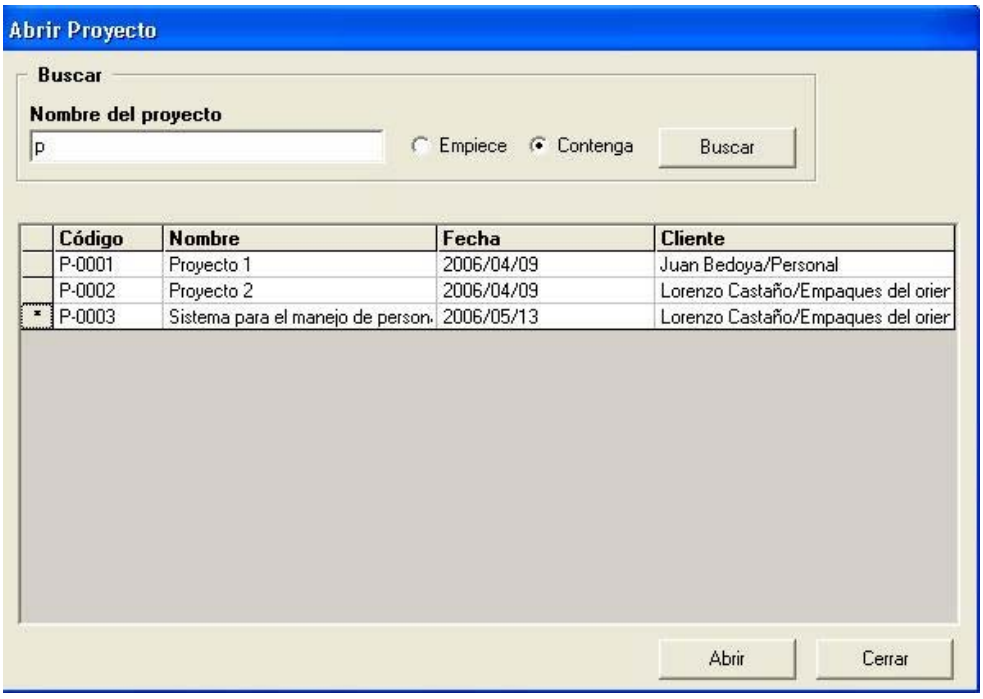
8.4.1.2.1 Menú Archivo

✓ **Nuevo Proyecto**. Esta opción permite la creación de un proyecto para empezar a realizar la especificación de requerimientos del mismo. Antes de elegir esta opción el cliente correspondiente al proyecto debe estar creado.



En el tab Modificar, se encuentran las opciones para modificar los datos ingresados al proyecto creado que pertenezcan al usuario que está utilizando la aplicación.

✓ **Abrir Proyecto.** En esta opción, el usuario podrá buscar los proyectos que estén a su cargo, ingresando como parámetro de búsqueda alguna palabra que pertenezca al nombre del proyecto que necesita abrir. Si el usuario no está seguro sobre qué palabra ingresar, puede dejar esta casilla en blanco y seleccionar la opción Buscar, que listará todos los proyectos de ese usuario en ese momento.



Luego, el usuario seleccionará el proyecto y la opción Abrir para cargar el proyecto al sistema y habilitar las opciones que dependan de este proceso.

- ✓ **Cerrar Sesión.** Con esta opción se vuelve a la interfaz de acceso.

- ✓ **Salir.** Es la opción para finalizar la aplicación.

8.4.1.2.2 Menú Administración

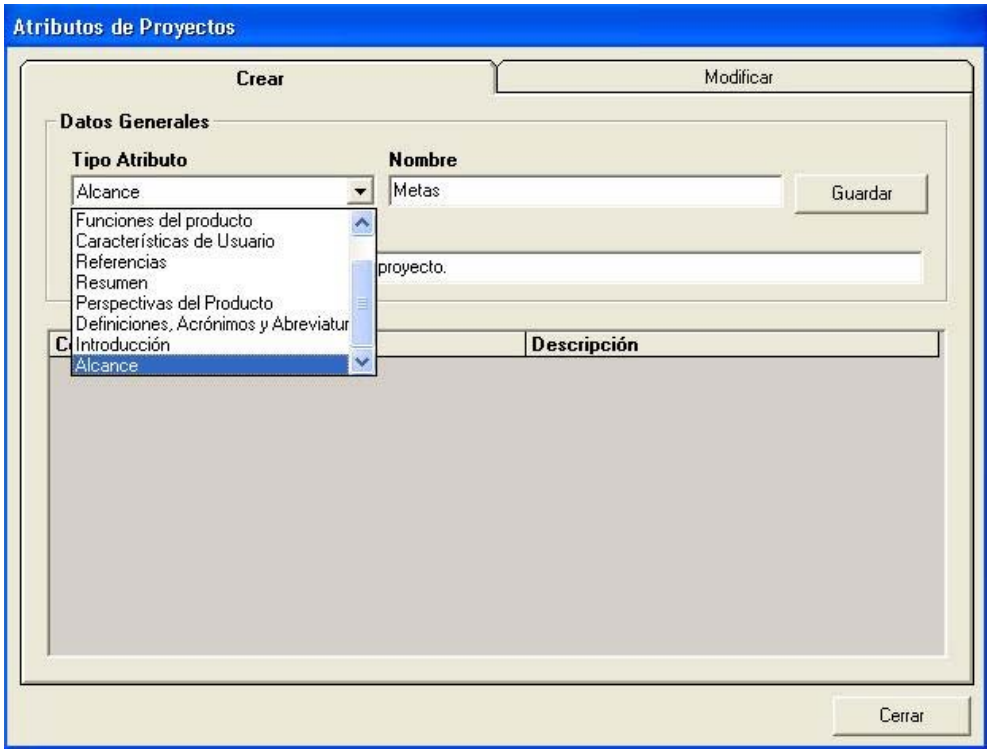
NOTA: Para una mejor compresión de la organización del proyecto, se recomienda leer el estándar IEEE 830 o en su defecto el apartado del capítulo 2 dedicado a la descripción de estándar.

- ✓ **Tipos de Atributos**. En esta opción se pueden crear, modificar o eliminar tipos de atributos de un proyecto. Estos atributos corresponden a ítems descriptores donde se define el alcance del proyecto, las definiciones, acrónimos y abreviaturas utilizadas, las referencias, entre otros. Si un tipo de atributo tiene registros relacionados en otras tablas no podrá ser eliminado.


Tipos de Atributos
Crear
Modificar
Datos Generales
Código
Nombre
10
Suposiciones y Dependencias
Guardar
Descripción
Lista cada uno de los factores que afectan los requisitos declarados en la SRS
Código
Nombre
Descripción
Cerrar


Tipos de Atributos
Crear
Modificar
Código
Nombre
Descripción
Eliminar
1
Introducción
2
Alcance
3
Referencias
4
Resumen
5
Perspectivas del Producto
6
Funciones del producto
7
Características de Usuario
8
Datos Generales
Código
Nombre
1
Introducción
Modificar
Descripción
Cerrar

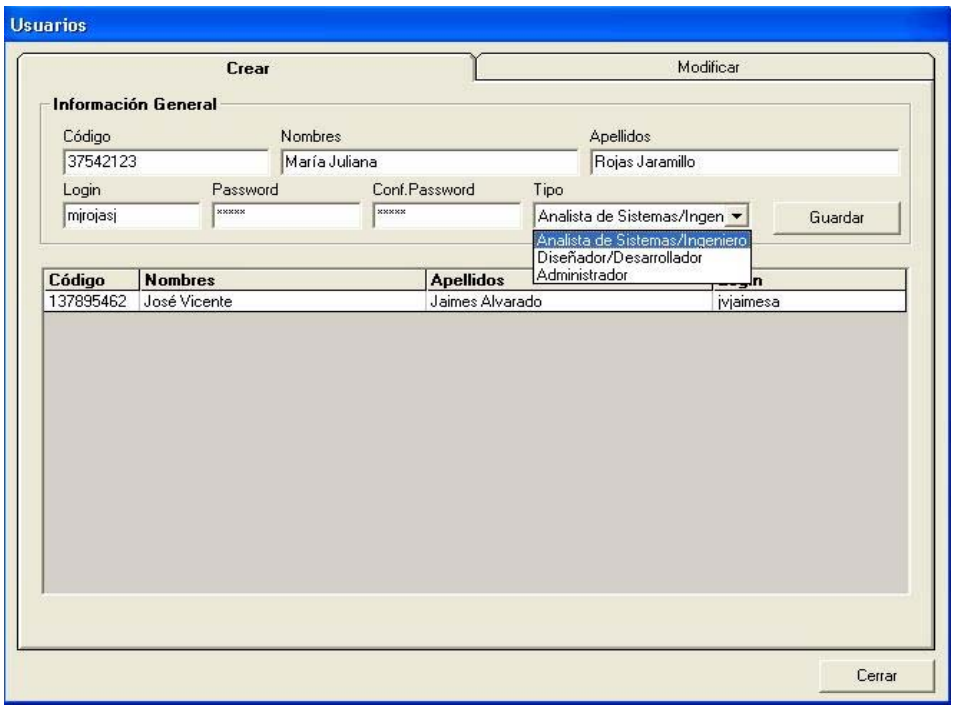
- ✓ **Atributos**. Esta opción permite crear, modificar o eliminar “atributos” de los tipos de atributos de un proyecto. Es decir estos atributos pertenecen a una clasificación establecida anteriormente como tipo de atributo. Por ejemplo, si se creó un tipo de atributo llamado “Alcance”, los atributos para éste tipo de atributo serán: Nombre del proyecto, beneficios obtenidos, objetivos, metas, entre otros. Si un atributo tiene registros relacionados en otras tablas no podrá ser eliminado.



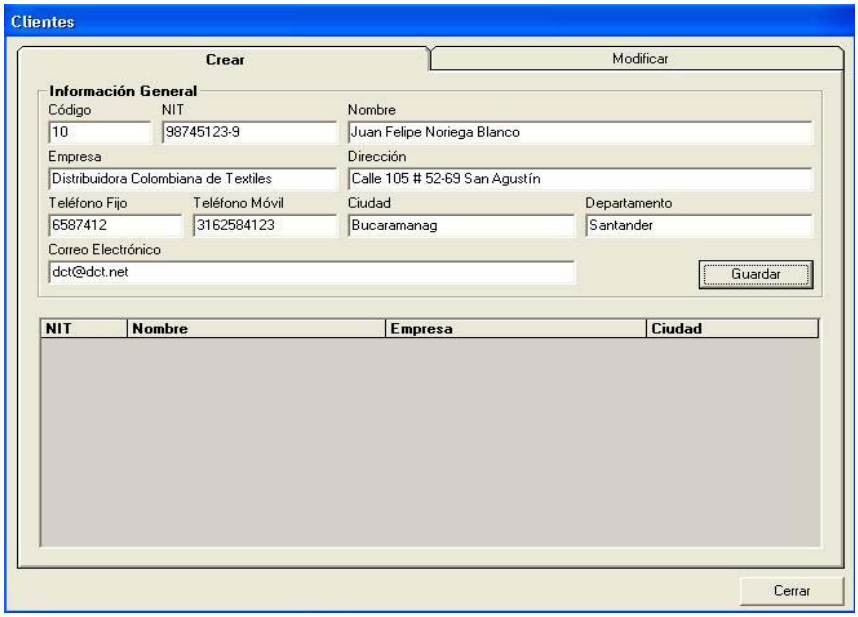
- ✓ **Tipos de Requerimientos**. En esta opción se pueden crear, modificar o eliminar tipos de requerimientos de un proyecto. Estos tipos de requerimientos corresponden a clasificaciones generales utilizadas para la especificación de requerimientos de un proyecto, por ejemplo, requerimientos funcionales, de rendimiento, de diseño, entre otros. Si un tipo de requerimiento tiene registros relacionados en otras tablas no podrá ser eliminado.

- ✓ **Atributos de Tipos de Requerimientos**. En esta opción se pueden crear, modificar o eliminar atributos que pertenezcan a los tipos de requerimientos definidos en la opción anterior. Por ejemplo, si se creó un tipo de requerimiento llamado "Requerimientos de Interfaces Externas", los atributos correspondientes que se crearían en esta opción serían Interfaces de usuario, interfaces de hardware, interfaces de software, etc. Si un tipo de requerimiento tiene registros relacionados en otras tablas no podrá ser eliminado.

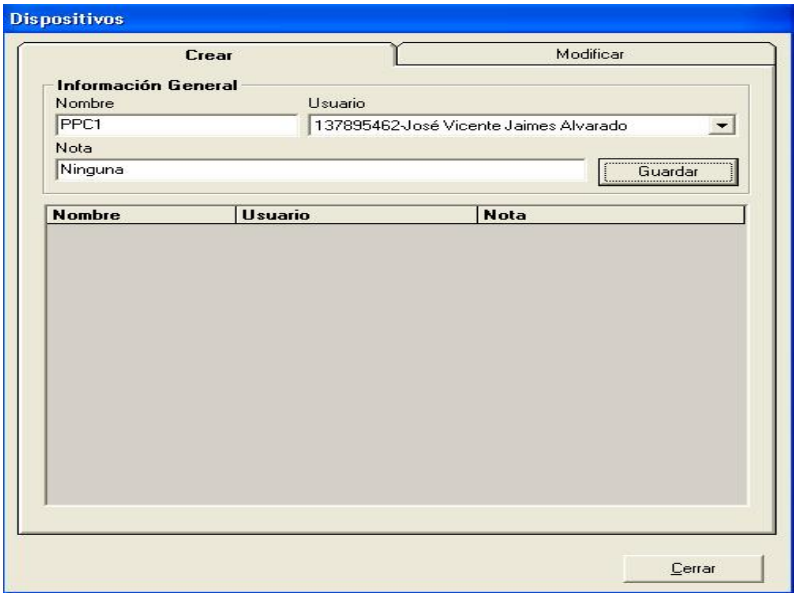
- ✓ **Usuarios**. Es donde se crean, modifican o eliminan los usuarios que van a utilizar la aplicación.



- ✓ **Clientes**. En esta opción se crean, modifican o eliminan los clientes a los cuales se les va a asociar un proyecto software.



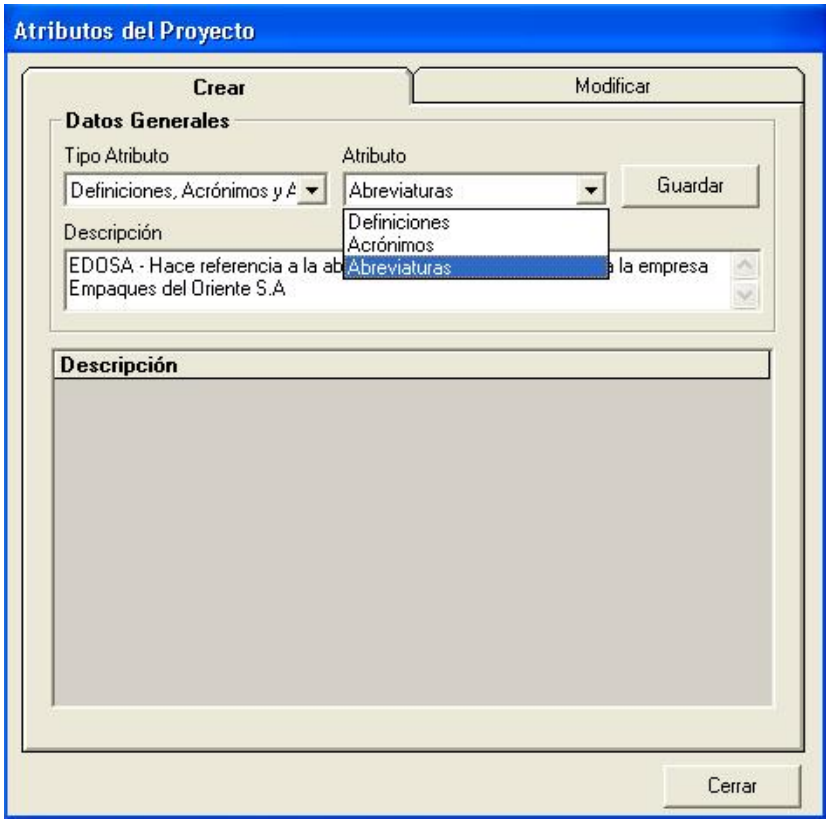
- ✓ **Dispositivos**. Es donde se crean, modifican o eliminan los dispositivos que van a ser asignados a los usuarios que van a utilizar la aplicación móvil.



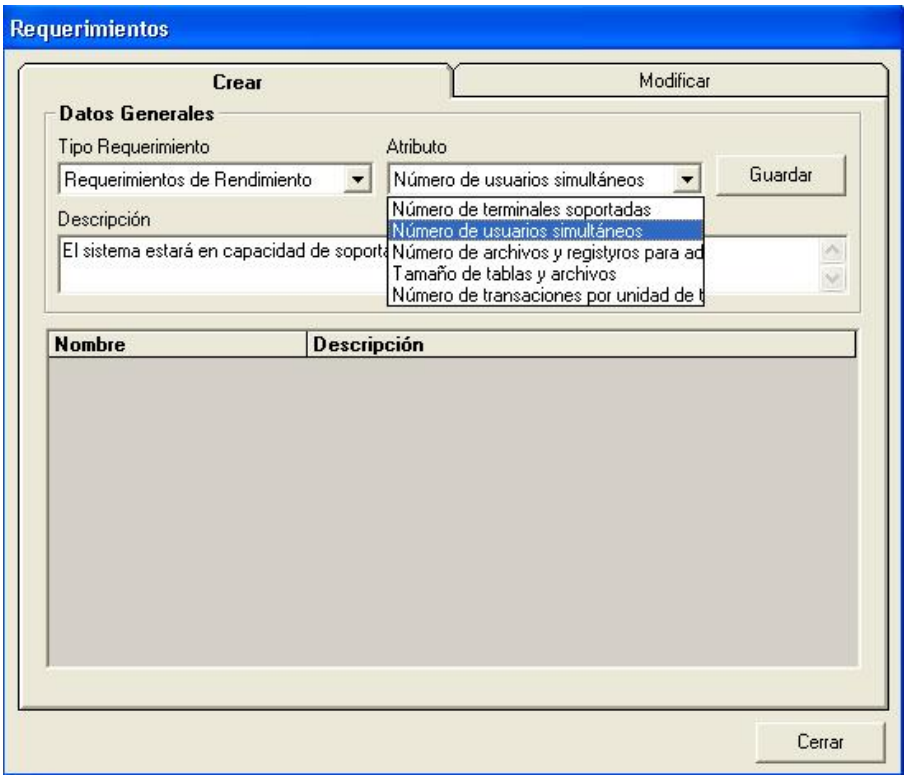
8.4.1.2.3 Menú Proyectos

A través de éste menú se pueden registrar los atributos y requerimientos de un proyecto determinado.

- ✓ **Atributos del Proyecto**. En esta opción se pueden crear, modificar o eliminar atributos de un proyecto software a partir de los tipos de atributos definidos en el menú Administración.



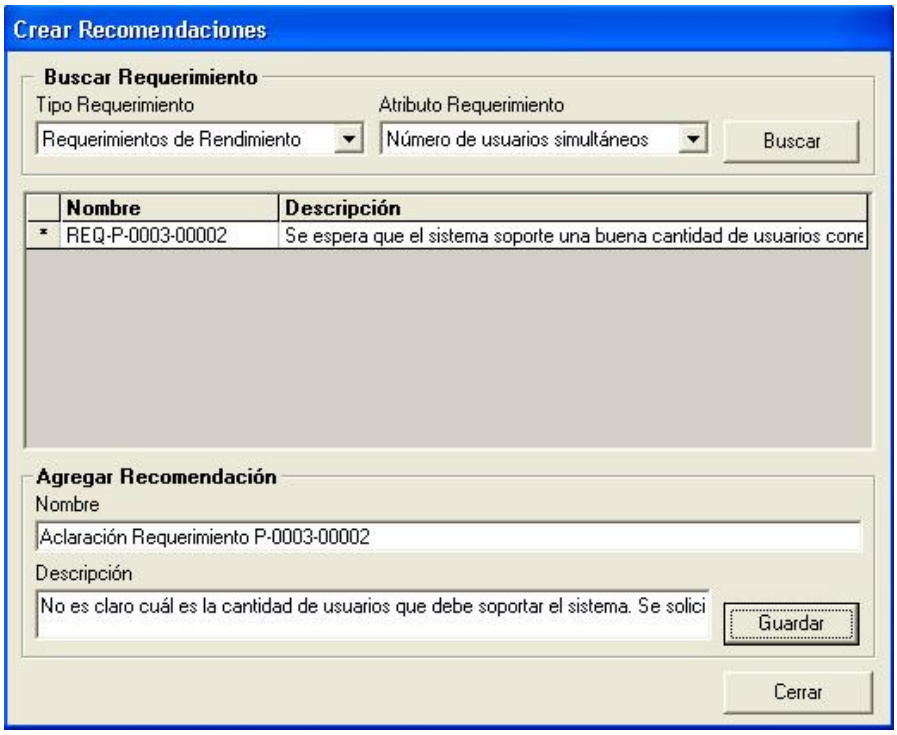
- ✓ **Requerimientos del Proyecto**. En esta opción se pueden crear, modificar o eliminar requerimientos de un proyecto software a partir de los tipos de requerimientos definidos en el menú Administración.



8.4.1.2.4 Menú Herramientas

- ✓ **Generar informe SRS**. En esta opción se puede generar el documento de especificación de requerimientos de software (SRS) del proyecto activo en formato RTF. El sistema informará la ruta en donde quedó guardado el documento.

- ✓ **Recomendaciones**. A esta opción pueden tener acceso los ingenieros que trabajan en un proyecto software, pero que no están encargados de la especificación de requerimientos del mismo. A través de ésta opción se pueden crear nuevas recomendaciones a los requerimientos definidos dentro de un proyecto. El usuario debe seleccionar el tipo de requerimiento del cual tiene inquietud e ingresar la descripción de la sugerencia a realizar para poder guardarla. También podrá consultar las recomendaciones realizadas a ese proyecto con anterioridad.



8.4.1.2.5 Menú Windows Mobile

Desde este menú se puede realizar la sincronización de datos entre el dispositivo móvil (Pocket PC) y la aplicación de escritorio. Para esto, se debe tener activa la aplicación ActiveSync y seleccionar la opción sincronizar dispositivos del menú Windows Mobile.

8.4.1.2.6 Menú Ayuda

En éste menú se encuentran los manuales de usuario para la aplicación de escritorio y móvil, además de la ventana Acerca de… con información básica de la aplicación.

8.4.2. Manual de Usuario Aplicación Móvil

8.4.2.1 Inicio de la Aplicación

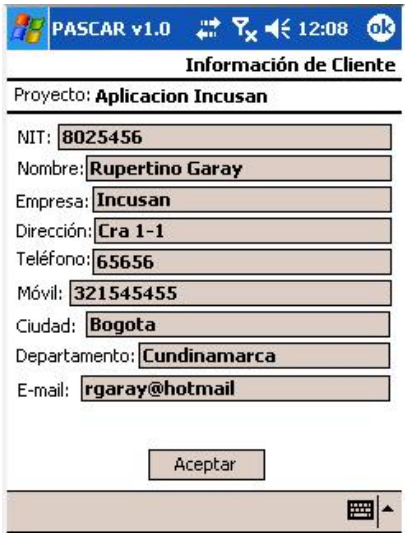
Cuando se inicia la aplicación móvil de PASCAR, aparece la ventana de autenticación de usuarios, donde se debe ingresar la contraseña (password) del usuario asignado al dispositivo. Se validará que la contraseña para el acceso a la aplicación.

8.4.2.2 Interfaz Principal

La interfaz principal muestra el listado de los proyectos de software a cargo del usuario de la aplicación móvil y tiene la opción de Búsqueda por proyectos. El usuario debe seleccionar el proyecto del cual desea obtener información.

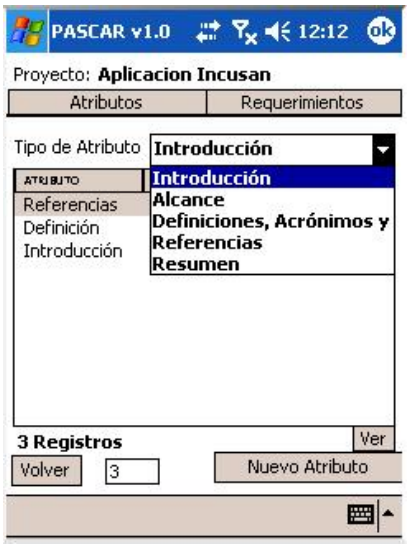
En esta interfaz encontramos las siguientes opciones para cada proyecto:

- ✓ **Opción "Ver".** Permite ver los datos básicos del cliente del proyecto seleccionado

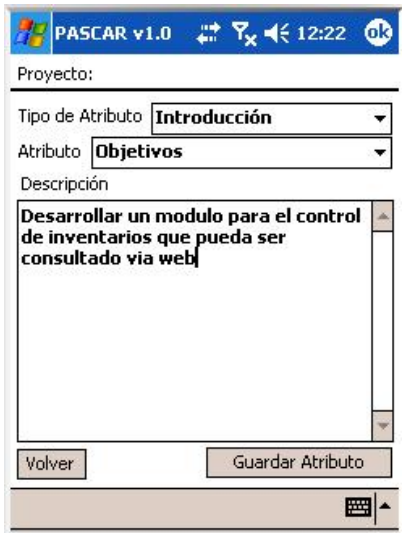


- ✓ **Opción "Todos".** Permite listar todos los proyectos disponibles en el dispositivo.

- ✓ **Opción "Ir a Proyecto".** Permite ingresar a la interfaz de atributos y requerimientos del proyecto seleccionado.



- ✓ **Opción "Nuevo Atributo/Nuevo requerimiento".** En esta interfaz podemos seleccionar "Nuevo Atributo" o "Nuevo Requerimiento", dependiendo de la opción que se haya seleccionado antes, para agregar nuevos atributos o requerimientos al proyecto en caso de ser necesario.



Los botones que tienen la etiqueta "Volver" retornan la aplicación a la interfaz anterior.

- ✓ **Opción "Salir".** Para salir de la aplicación se debe seleccionar la opción "Salir" que se encuentra en la interfaz principal.