UNIVERSIDAD CATOLICA SANTA MARIA

FACULTAD DE CIENCIAS E INGENIERIAS FÍSICAS Y FORMALES

ESCUELA PROFESIONAL DE INGENIERIA DE SISTEMAS

PROPUESTA DE UNA TÉCNICA PARA IMPLEMENTAR PRUEBAS DE SOFTWARE EN EL CICLO DE DESARROLLO DEL SOFTWARE UTILIZANDO PMBOK EN LAS COOPERATIVAS DE AHORRO Y CRÉDITO

Tesis presentado por el Bachiller:

MARCO ANTONIO ESCOBEDO MONROY

Para optar el Título Profesional de: INGENIERO DE SISTEMAS

Asesor:

Ing. Fernando Paredes Marchena

AREQUIPA-PERÚ

2017

PRESENTACIÓN

Sr. Director del Programa Profesional de Ingeniería de Sistemas.

Srs. Miembros del Jurado.

De conformidad con las disposiciones del Reglamento de Grados y Títulos del Programa Profesional de ingeniería de Sistemas, ponemos a vuestra consideración el presente trabajo de investigación titulado:

"PROPUESTA DE UNA TECNICA PARA IMPLEMENTAR PRUEBAS DE SOFTWARE EN EL CICLO DE DESARROLLO DEL SOFTWARE UTILIZANDO PMBOK EN LAS COOPERATIVAS DE AHORRO Y CREDITO", el mismo que de ser aprobado me permitirá optar el Título Profesional de Ingeniería de Sistemas.

Marco Antonio Escobedo Monroy

A Dios, mi amada esposa Coralea mis hijos Thadeo, Esteban y Sergio por su cariño y comprensión a mis queridos padres Emigdio y Julia que siempre han estado a mi lado apoyándome y dándome fuerzas para cumplir cada uno de mis sueños y éste es uno de ellos; a ustedes por siempre mi amor y Agradecimiento

RESUMEN

Para construir productos de software se emplean modelos, métodos, metodologías y técnicas que respalden los procesos del desarrollo de software. Estas actividades le insertan características para una buena construcción asegurándonos buenos resultados. Durante la misma se utilizan un conjunto de pruebas de software que se convierten en una ayuda en cualquiera de las fases de elaboración. Es común que se empleen estas pruebas de software sin criterios tomándose solo en cuenta la funcionalidad del producto.

El presente trabajo de investigación propone una técnica donde se presentan un conjunto de criterios que permitan seleccionar pruebas de software durante el ciclo de desarrollo del mismo. Asimismo, permite el proceso una retroalimentación para las fases de análisis, diseño y construcción del software basado en la experiencia de los constructores.

La definición de las pruebas de software para cada una de las fases de desarrollo del producto es importante ya que no se espera a la culminación del producto para analizar su trazabilidad con los requerimientos.

Palabras clave: Pruebas de software, modelo de calidad, proyecto de software, criterios de selección.

ABSTRACT

Software products to build models, methods, methodologies and techniques to support the software development processes are employed. These activities will insert features a good construction ensuring good results. During the same set of software testing that become an aid in any stage of development they are used. It is common for these software testing criteria are used without taking into account only the functionality of the product.

This paper proposes a technique where a set of criteria to select software testing during the development cycle thereof are presented. It also allows the process feedback to the phases of analysis, design and construction based on the experience of the builders software.

The definition of software testing for each of the phases of product development is important because it is not expected to completion of the product to analyze the requirements traceability.

Keywords: software testing, quality model, software project selection criteria.

INTRODUCCION

Las pruebas de software desempeñan un importante papel en la construcción de productos de software y en la determinación si el producto hace lo que el cliente desea. Como consecuencia del avance metodológico, las pruebas de software han experimentado continuos cambios convirtiéndose en verdaderas soluciones para lograr productos adecuados.

En la actualidad, las pruebas de software definidas están siendo documentadas y en donde evidencian su utilidad; muchos trabajos de investigación revelan grandes avances en la concepción de estas pruebas. Obviamente, existe literatura técnica explicando la forma como pueden ser definidas estas e identificando la necesidad de canalizar iniciativas para formular pruebas específicas que apoyen la construcción de buenos productos de software.

Este trabajo de investigación, formula un conjunto de criterios, aglutinadas en una técnica que identifican los tópicos relevantes para el uso de las pruebas de software y su relación con el ciclo de vida de construcción. Así, en el capítulo 1 se presenta el planteamiento operacional; en el capítulo 2 se muestra los antecedentes de investigación y el marco teórico. El capítulo 3 presenta la descripción de la propuesta en forma detallada para que en el capítulo 4 se presente un caso de estudio práctico. El capítulo 5 valida la propuesta en base a la opinión de expertos; finalmente se presentan las conclusiones y recomendaciones del caso.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

1.1 Título del proyecto

PROPUESTA DE UNA TECNICA PARA IMPLEMENTAR PRUEBAS DE SOFTWARE EN EL CICLO DE DESARROLLO DEL SOFTWARE UTILIZANDO PMBOK EN LAS COOPERATIVAS DE AHORRO Y CREDITO

1.2 Descripción del problema

El objetivo de la Ingeniería de Software es construir un producto de software o mejorar alguno existente. Un proceso de desarrollo de software efectivo proporciona normas para la construcción eficiente de software de calidad. Este proceso está compuesto, a grandes rasgos, por cinco etapas: Requisitos, Análisis, Diseño, Implementación y Prueba.

Uno de los objetivos de la etapa de diseño es crear una abstracción de la implementación, en el sentido de que la implementación como un refinamiento directo del diseño. Esto permite utilizar tecnologías como la generación de código y la ingeniería de ida y vuelta entre el diseño y la implementación, entre otros. Es por esto que el diseño está directamente relacionado con los lenguajes de programación y es en ésta etapa, donde se construyen modelos que dependen de ellos.

Por otro lado, uno de los principales problemas en el proceso de desarrollo de software es el uso de grandes cantidades de recursos. Aún con el incremento de la automatización de diferentes actividades del desarrollo de software, los recursos

son escasos y costosos. Como consecuencia de esto y en la constante búsqueda de mejoras en el desarrollo de sistemas de software, los investigadores y desarrolladores están utilizando varias orientaciones en el proceso de desarrollo (Somerville, 2007).

Como se menciona la industria del software ha atravesado una serie de etapas a lo largo del tiempo desde sus inicios hasta llegar al desarrollo rápido de calidad requerido en los noventa. La calidad de software es un punto crucial en cualquier desarrollo serio. Para determinar si un software es de calidad o no se pueden utilizar métricas de software las cuales son también aplicables a otras áreas como prueba, administración y mantenimiento del software.

En los últimos años existe un gran interés en las métricas de software debido a su potencial para predecir el uso más eficiente de los recursos y mejorar la calidad. Se han realizado una gran cantidad de trabajos relacionados a las métricas, en especial aplicadas a la orientación a objetos y procedimentales (Dines, 2006) (Lorenz et al, 1994), pero hay muy pocos trabajos realizados con métricas específicamente para sistemas donde se integran las pruebas de software al ciclo de desarrollo del mismo. Algunos investigadores han establecido que si se aplican metodologías para el desarrollo se mejoran los valores obtenidos en las métricas para la orientación a objetos (Zhao, 2008).

Se sabe que las métricas de software son formas de calificar el diseño de software. Se puede decir entonces que las métricas aplicadas son cruciales para determinar la efectividad de su uso. Tanto el diseño como el modelado de

sistemas ocupan un lugar importante en la Ingeniería de Software.

Los modelos brindan la posibilidad de abstraer sistemas y facilitar la implementación. Por otra parte, la construcción de buenos modelos asegura un correcto desarrollo de la arquitectura del sistema. Es imprescindible entonces contar con herramientas que permitan evaluar la calidad del diseño, es por esto que la definición de métricas específicas es indispensable.

La definición de métricas permitirá evaluar objetivamente el desarrollo de un producto de software de manera tal que se logre aprovechar al máximo las ventajas. Considerando lo antes expuesto, se implementará la definición de métricas específicas para sistemas, aplicables a partir de la etapa del diseño. En general es en esta etapa donde surge el modelado de los aspectos del sistema. Uno de los inconvenientes encontrados al momento de definir métricas, para el diseño, es que en la actualidad no hay acuerdo con respecto a la integración de las pruebas de software al ciclo de desarrollo del mismo.

1.3 Definición del problema

La Ingeniería de Software, busca producir una descripción detallada de un problema, con el fin de construir un Sistema de Software, que satisfaga las "necesidades y objetivos" de la organización donde funcionará dicho sistema con criterios de construcción de calidad del producto. En la comunidad de Ingeniería de Software, estos objetivos constituyen el fundamento del sistema, y son usualmente definidos como las metas a ser cumplidas por el sistema y su entorno, aunque algunos autores distinguen los objetivos del sistema de los objetivos de la

organización.

El estudio de las etapas críticas es útil porque permite establecer cronogramas y costos precisos que es uno de los desafíos más relevantes para el planeamiento de proyectos de software. La contribución de insertar pruebas de software en las etapas de análisis y diseño puede asegurarnos estar cerca de productos bien elaborados; y establecer las formas y elementos que permitan un conocimiento exhaustivo de la situación. Esto permite asegurarnos del cumplimiento de los objetivos y de los planes establecidos.

Para lograr la calidad del producto se debe de establecer medidas como por ejemplo el alcance calificado, el porcentaje de clicks, la percepción del reconocimiento del producto, el resultado del trabajo, la calificación final de acción y la eficiencia; logrando dar respuesta a las siguientes razones para medir: Caracterizar, evaluar, predecir y mejorar. Para obtener todos estos datos, se debe de avaluar el producto en el momento de su producción y lograr medidas que permitirán mejores ajustes en proyectos futuros.

Aplicado la técnica se espera obtener elementos que permitan cuantificar, en versiones tempranas, la calidad del producto a obtener. Esto permitirá entender los niveles de satisfacción de los constructores a medida que el producto se construye.

El presente trabajo de investigación permite profundidad en el tema de la calidad de software que actualmente forma parte de las características esperadas de un

software al entregarse a un cliente. El hecho de conocer lo relacionado con las pruebas de software, su funcionamiento y su posible ciclo acoplándose con técnicas y metodologías de punta ayudan a tener una idea más clara acerca de cómo se deben plantear entornos de alta calidad en empresas que construyan y reciban software.

Los inconvenientes que se pueden encontrar es que no se encuentren correctamente definidos los escenarios que permitan una definición clara de la aplicación de las pruebas de software. Ante este inconveniente, es necesario elaborar un proceso para que a partir de las actividades permita atacar de manera adecuada las etapas críticas respectivamente.

1.4 Formulación del problema

Encontrar una técnica que permita escalar, formar e integrar pruebas de software durante las etapas de construcción del producto.

1.5 Objetivos de la investigación

1.5.1 Objetivo general

Proponer una técnica que permita implementar pruebas de software en el ciclo de desarrollo del software empleando PMBOK y para Cooperativas de Ahorro y Crédito.

1.5.2 Objetivos específicos

- Diseñar una técnica de escalamiento, empleando PMBOK, que permita detectar las causas a los problemas de las pruebas del software.

- Optimizando métricas de calidad para reconocer el estado de las prácticas del ciclo de desarrollo de software.
- Desarrollar un proceso de soporte para la formación de pruebas de software basados en PMBOK.

1.6 Viabilidad de la investigación

1.6.1 Económica

Se cuenta con los recursos económicos necesarios para solventar el presente trabajo de investigación.

1.6.2 Técnica

Se cuenta con la capacidad académica para resolver el problema.

1.6.3 Operativa

Medios bibliográficos, Internet, bibliotecas y otras universidades con infraestructura, laboratorios, entre otros.

1.7 Justificación e importancia de la investigación

1.7.1 Justificación

La Ingeniería de Software, busca producir una descripción detallada de un problema, con el fin de construir un Sistema de Software, que satisfaga las "necesidades y objetivos" de la organización donde funcionará dicho sistema. En la comunidad de Ingeniería de Software, estos objetivos constituyen el fundamento del sistema, y son usualmente definidos como las metas a ser cumplidas por el sistema y su entorno,

aunque algunos autores distinguen los objetivos del sistema de los objetivos de la organización.

El estudio de las etapas críticas es útil porque permite establecer cronogramas y costos precisos que es uno de los desafíos más relevantes para el planeamiento de proyectos de software. La contribución de insertar pruebas de software en las etapas de análisis y diseño asegura estar más cerca de lo deseado y comprometido; y establecer las formas y elementos que permitan un conocimiento exhaustivo de la situación. Esto permite asegurarnos del cumplimiento de los objetivos y de los planes establecidos. Para lograr la calidad del producto se debe de establecer medidas como por ejemplo el alcance calificado, el porcentaje de clicks, la percepción del reconocimiento del producto, el resultado del trabajo, la calificación final de acción y la eficiencia; logrando dar respuesta a las siguientes razones para medir: Caracterizar, evaluar, predecir y mejorar. Para obtener todos estos datos, se debe de avaluar el producto en el momento de su producción y lograr medidas que permitirán mejores ajustes en proyectos futuros.

1.7.2 Importancia

Permitir a los desarrolladores entender la forma cómo y cuándo se pueden escalar, formar e integrar pruebas de software en la gestión de proyectos de desarrollo de software.

1.8 Alcances y Limitaciones

1.8.1 Alcances

- Reducir el tiempo de construcción de productos de software al insertar pruebas de software en etapas tempranas de la construcción del mismo.
- Elaborar una técnica de escalamiento, que permita detectar los problemas antes de concluir con la construcción del producto.
- Proponer métricas de calidad que permitan reconocer el estado de la construcción del producto de software.

1.8.2 Limitaciones

El trabajo de investigación no contempla a aquellas pruebas de software relacionadas con la complejidad ciclomática. Asimismo, no se contempla emplear metodologías ágiles que generan otro tipo de artefacto usado en la construcción del producto de software.

1.8.3 Delimitaciones

a. Delimitación espacial

El presente trabajo de investigación se lleva a cabo en la ciudad de Arequipa.

b. Delimitación temporal

El trabajo se inicia en julio del 2012 y culmina en setiembre del 2013.

c. Delimitación social

Está orientado a resolver problemas de construcción de software, lo cual implica a Ingenieros de Sistemas, Ingenieros Informáticos, Licenciados en Ciencias de la Computación, Ingenieros de Software, Ingenieros de Requisitos y Arquitectos de Software.

d. Delimitación conceptual

Gestión de Proyectos de Software, Pruebas de Software.

1.9 Hipótesis de la investigación

1.9.1 Hipótesis general

Dada la importancia que representan las pruebas de software durante la gestión de los proyectos de desarrollo de software concluidos, es probable que a partir de la gestión de pruebas de software, la concepción de nuevos proyectos de desarrollo de software permita proporcionar confianza en su construcción.

1.10 Variables e indicadores

1.10.1 Variable independiente

- Pruebas de software
- PMBOK.

1.10.2 Variable dependiente

- Técnica para implementar pruebas de software.

1.10.3 Indicadores

1.10.3.1 Variables independientes

- Usabilidad
- Eficiencia
- Facilidad

1.10.3.2 Variables dependientes

- Confiabilidad

1.11 Área, línea, tipo y nivel de investigación

1.11.1 Área de investigación

El área de investigación es la de Ingeniería de Software.

1.11.2 Línea de investigación

La línea de investigación es la de Gestión de Pruebas de Software.

1.11.3 Tipo de investigación

Documentada de campo porque se desea obtener un conjunto de soluciones de software por medio de la revisión de un conjunto de productos de software ya construidos.

1.11.4 Nivel de investigación

Exploratorio porque la concepción de la gestión de etapas críticas de los proyectos de software se encuentra en una fase insípida, descriptiva

porque se pretende realizar un análisis del estado del objeto de estudio y experimental porque se intenta ensayar una nueva solución con la finalidad de ayudar a resolver situaciones concretas en la calidad de los productos de software.

1.12 Cobertura del estudio

1.12.1 Universo

Uno de los mayores problemas que presenta nuestra ciudad es la confección de estadísticas sobre el área de tecnología, a pesar de los esfuerzos que viene realizando el gobierno peruano por medio del Instituto de Estadística e Informática, no se ha podido consolidar esta información por lo que nuestro universo se encuentra conformado por personas dedicadas al desarrollo de Software que se encuentren trabajando actualmente y que empleen las técnicas de desarrollo en su totalidad.

1.12.2 Muestra

Al no existir estadísticas o métricas sobre información referente a la construcción de software y a todos sus activos; se realiza un muestreo por cuotas. Se visita a las empresas dedicadas a ésta área y se considera a las personas orientadas a estos menesteres, que hayan trabajado y que trabajen con la documentación correcta en la construcción de software.

1.13 Técnicas e instrumentos de recolección de información

1.13.1 Técnicas

- Entrevista
- Cuestionario
- Revisión bibliográfica
- Revisión documental

1.13.2 Instrumentos

- Entrevista personal
- Encuesta
- Textos
- Documentos propios de la organización

2.1 Antecedentes investigativos

(Castillo et al, 2000) presenta un compendio de los conceptos de administración aplicada a proyecto, y se amplía con teoría pertenecientes a otras áreas importantes que deben tomarse en cuenta para la conclusión eficiente de un proyecto de software. Además trata de la importancia que han tomado estas definiciones en un mundo cambiante con tendencias globales; la herramienta es útil para las diferentes empresas e instituciones que por las exigencias en su búsqueda por la excelencia y la globalización, han tenido que incorporarlas a sus empresas, con el propósito de mantenerse como líder en el mercado.

El éxito o fracaso del proyecto depende en gran medida del seguimiento adecuado que se da a los problemas que se presentan y el tiempo en que estos sean resueltos. El los proyectos de software solo se puede estar seguro de algo: Que se tendrán problemas. Pero esto no debe crear miedo sino todo lo contrario, se debe de tener una mente abierta para enfrentarlos y en el mejor de los casos prevenirlo. En el proyecto de investigación se explican las metas, objetivos, alcance del proyecto, trabajo en equipo, el apoyo de la Gerencia, el compromiso del equipo de trabajo, el uso de motivadores, y otras herramientas necesarias para obtener resultados satisfactorios y cercanos a los planeados en una Administración de Proyectos de Software.

(Montiel, 2006) menciona que las organizaciones sociales o empresas van creciendo con el desarrollo tecnológico y tratan de adaptarse a los cambios del entorno. Bajo esta dinámica, el flujo de la información es parte importante para

el crecimiento organizacional y la aplicación de proyectos en las empresas se ha convertido en un factor importante para su propio desarrollo. Para garantizar proyectos exitosos que beneficien a las empresas u organizaciones, se deben tomar en cuenta varios factores que impacten en la realización e implantación de los proyectos.

(Diez, 2003) indica que una de las formas de adaptar una metodología estándar, a un proyecto en particular, es a través del mapa de actividades para ese proyecto, dadas las características del mismo. La confección del mapa de actividades no es una tarea trivial ni automática, para hacerla correctamente se requiere de capacidad de análisis, conocimientos y experiencia en la aplicación de metodologías estándares de desarrollo. Se presenta en esta tesis un sistema basado en conocimientos que asiste al responsable de un proyecto de desarrollo de software, en la elaboración del mapa de actividades del mismo. El sistema permite ingresar las particularidades del proyecto, infiere el mapa de actividades sobre la base de la metodología estándar Métrica Versión 3 y lo presenta en un formato electrónico estándar.

(Marante, 2009) hace hincapié que un proyecto de desarrollo y/o mantenimiento de software conlleva todas las dificultades de un proyecto de ingeniería, pero además incluye los particulares retos que tiene la construcción de un producto software; complejidad de la implementación, requisitos volátiles, desafíos tecnológicos, desarrollo colaborativo, dificultad para asegurar la calidad, etc. Además, el mercado cada vez exige plazos de entrega más reducidos y presupuestos más ajustados. Las empresas de desarrollo de software buscan dar

respuestas a estos retos mediante procesos que se centren en mejorar la productividad y calidad del desarrollo, de forma que ayuden a cumplir con sus compromisos en los plazos de entrega establecidos.

Los proyectos de desarrollo de software suelen enfrentarse a un ámbito de trabajo muy cambiante (especialmente en cuanto a las especificaciones del producto y las prioridades de las características solicitadas) e intensivo en comunicación (entre el equipo y con el cliente). Las técnicas y herramientas genéricas para seguimiento de proyectos resultan claramente ineficaces para enfrentar estos desafíos. Las metodologías ágiles destacan esta situación pero la resuelven de una manera excesivamente simplista, utilizando roles muy genéricos (reduciendo así la comunicación necesaria entre diferentes roles) y confiando en la habilidad de cada uno de los miembros del equipo para que, verbalmente y/o con soportes no informatizados, realicen el seguimiento continuo del proyecto

2.2 Marco conceptual

2.2.1 Gestión de Proyectos de Software.

2.2.1.1 Planificación de un Proyecto de Ingeniería de Software.

La planificación involucra la especificación de objetivos y metas para un proyecto y las estrategias, políticas, planes y procedimientos para alcanzarlos. Todo proyecto de ingeniería de software debe partir con un buen plan. La planificación es necesaria por la existencia de incertezas sobre el ambiente del proyecto software y sobre fuentes externas. La planificación enfoca su atención en las metas del proyecto, riesgos

potenciales y problemas que puedan interferir con el cumplimiento de esas metas (McConnell, 1996). Los principales problemas en la planificación de un proyecto de ingeniería de software incluyen los siguientes (Pressmann, 2005) (Randolph, 1993):

- Requisitos incorrectos e incompletos.
- Muchas especificaciones de requisitos son inestables y sujetas a cambios mayores.
- La planificación no se lleva a cabo por la creencia errónea de que es una pérdida de tiempo y los planes cambiarán de todos modos.
- La planificación de costos y plazos no es actualizada y se basa en necesidades de mercadeo y no de los requisitos del sistema.
- Es difícil estimar el tamaño y complejidad del proyecto de software de modo de realizar una estimación de costos y plazos realista.
- Los costos y plazos no son reestimados cuando los requisitos del sistema o el ambiente de desarrollo cambia.
- No se manejan factores de riesgo.
- La mayoría de las organizaciones de desarrollo de software no recolectan datos de proyectos pasados.
- Las compañías no establecen políticas o procesos de desarrollo de software.

2.2.1.1.1 Actividades que se derivan de la planificación (Thayer, 1998).

- Fijar los objetivos y metas
- Desarrollar estrategias

- Desarrollar políticas
- Anticipar futuras situaciones
- Conducir un establecimiento de riesgos
- Determinar posibles cursos de acción
- Tomar decisiones de planificación
- Fijar procedimientos y reglas
- Desarrollar los planes del proyecto
- Preparar presupuestos
- Documentar los planes del proyecto.

2.2.1.2 Organización de un proyecto de Ingeniería de Software.

Involucra desarrollar una estructura organizacional efectiva y eficiente para asignar y completar las tareas del proyecto y establecer las relaciones de autoridad y responsabilidad entre las tareas (Thayer, 1998). Los principales problemas en la organización de un proyecto de ingeniería de software incluyen los siguientes (Varas, 1998):

- Es difícil determinar la mejor estructura organizacional para una organización y/o ambiente particular (por ejemplo tipo proyecto, funcional o matriz) para gestionar el proyecto.
- Una estructura organizacional puede dejar responsabilidades para algunas actividades y tareas del proyecto poco claras o indefinidas.
- Mucho personal de desarrollo de software no acepta una organización matricial.
- Muchos líderes de equipo esperan desarrollarse tanto técnicamente como en la gestión de su equipo de trabajo.

2.2.1.2.1 Actividades que se derivan de la organización.

- Identificar y agrupar las funciones, actividades y tareas del proyecto.
- Seleccionar estructuras organizacionales
- Crear posiciones organizacionales
- Definir responsabilidades y autoridades.
- Establecer el perfil de cada puesto
- Documentar las decisiones organizacionales

2.2.1.3 Consiguiendo personal para un proyecto de Ingeniería de Software.

Consiste en todas aquellas actividades que involucran llenar (y mantener llenos) los puestos que fueron establecidos en la estructura organizacional del proyecto. Esto incluye selección de candidatos, entrenamiento y otros (Thayer, 1998) (Varas, 1998). Los principales problemas en esta etapa son:

- Los jefes de proyecto son frecuentemente seleccionados por su habilidad para programar o realizar tareas de ingeniería en vez de su habilidad de gestión (pocos ingenieros son buenos gerentes)
- La productividad de los programadores, analistas e ingenieros de software varía mucho de individuo en individuo.
- Hay grandes cambios en el equipo de un proyecto software, especialmente en aquellos organizados matricialmente.
- Las universidades no están produciendo un número suficiente de ingenieros que entiendan el proceso de la ingeniería de software o gestión de proyectos.

- Los planes de entrenamiento para desarrolladores individuales de software no se desarrollan o mantienen.

2.2.1.3.1 Actividades derivadas:

- Llenar los puestos de la organización.
- Asimilar al personal recientemente asignado
- Educar o entrenar al personal
- Proveer de desarrollo general
- Evaluar y valorar al personal
- Compensar

2.2.1.4. Dirección de un proyecto de ingeniería de software.

Dirigir un proyecto de ingeniería de software consiste en aquellas actividades de gestión que involucran aspectos interpersonales y de motivación por medio de las cuales el personal del proyecto entiende y contribuye a alcanzar los objetivos del proyecto. Una vez que los subordinados son entrenados y orientados, el jefe de proyecto tiene una responsabilidad continua por clarificar sus asignaciones, guiándolos hacia la mejora de la productividad, y motivándolos a trabajar con entusiasmo y confianza hacia las metas del proyecto (Thayer, 1998) (Varas, 1998). Los principales problemas en la dirección son:

- Fallas para tener una comunicación efectiva entre las entidades del proyecto y aquellas que no pertenecen al proyecto.
- El dinero no es un motivador suficiente para los desarrolladores de software.

- Las compañías y los jefes no poseen las técnicas y herramientas apropiadas para motivar a los ingenieros de software.
- Los clientes y gerentes no reconocen el impacto potencial en el software causado por un aparentemente cambio trivial, por ejemplo, ellos creen que es “sólo un problema simple de programación”.

Actividades de dirección:

- Proveer liderazgo
- Supervisar personal
- Delegar autoridad
- Motivar personal
- Construir equipos
- Coordinar actividades
- Facilitar comunicaciones
- Resolver conflictos
- Manejar cambios
- Documentar las decisiones de dirección.

2.2.1.5 Control de un proyecto de ingeniería de software.

Controlar es el conjunto de actividades de gestión utilizadas para asegurar que el proyecto va de acuerdo a lo planificado. El desempeño y los resultados se miden contra los planes, se notan las desviaciones, y se toman acciones correctivas. El control es un sistema de retroalimentación que provee información acerca de cuán bien va el proyecto (Pressmann, 2005) (Sommerville, 2005). El control responde las preguntas:

- ¿Está el proyecto en itinerario?

- ¿Está dentro de los costos?
- ¿Existen problemas potenciales que causen retrasos en alcanzar los requisitos dentro del presupuesto y plazo?

Los principales problemas en el control son:

- Muchos métodos de control de proyectos de desarrollo de software confían en los gastos del presupuesto para medir el "progreso" sin considerar el trabajo que lo acompaña.
- La visibilidad del progreso en un proyecto de software es difícil de medir.
- La calidad no es requerida, monitoreada o controlada.
- A menudo los estándares para el desarrollo de software no están escritos o, si lo están, no se fuerzan.
- El cuerpo de conocimiento llamado métricas de software (usadas para medir productividad, calidad, y progreso de un producto software) no está completamente desarrollado.

Actividades de control:

- Desarrollar estándares de desempeño
- Establecer sistemas de monitoreo y reportes
- Medir y analizar resultados
- Iniciar acciones correctivas
- Recompensar y disciplinar
- Documentar los métodos de control.

2.2.1.6 Los problemas y errores comunes.

Los desarrolladores, directivos y clientes normalmente tienen buenas razones para tomar las decisiones que toman, y la apariencia seductora de los errores clásicos es una de las razones de que esos errores se cometan tan a menudo.

Pero debido a que se han cometido muchas veces, sus consecuencias se han hecho fáciles de predecir. Y los errores rara vez producen los resultados que la gente espera (Pressmann, 2005) (Sommerville, 2005).

2.2.1.6.1 Personas.

A continuación aparecen algunos de los errores clásicos relacionados con las personas (Pressmann, 2005) (Sommerville, 2005).

- Motivación débil. Estudio tras estudio ha mostrado que la motivación probablemente tiene mayor efecto sobre la productividad y la calidad que ningún otro factor. Ejemplo: directivos que a lo largo de todo el proyecto toman medidas que minan la moral: como dar ánimos a diario al principio para pedir horas extras en la mitad, y como irse de vacaciones mientras el equipo está trabajando incluso los días de fiesta, para dar recompensas al final del proyecto que resultan ser de menos de un dólar por cada hora extra.
- Personal mediocre. Después de la motivación, la capacidad individual de los miembros del equipo, así como sus relaciones como equipo, probablemente tienen la mayor influencia en la

productividad. Contratar apurando el fondo del barril supondrá una amenaza al desarrollo. Ejemplo, hacer la selección del personal buscando quién puede contratarse más rápido, en vez de quién realizará la mayoría del trabajo durante la vida del proyecto. Esta técnica consigue un inicio rápido del proyecto, pero no determina un final rápido.

- Empleados problemáticos incontrolados. Un fallo al tratar con personal problemático también amenaza la velocidad de desarrollo. Un fallo al tomar una decisión cuando se trata con un empleado problemático es una de las quejas más comunes que tienen los miembros del equipo respecto de sus responsabilidades. Ejemplo, el equipo sabe que uno de ellos es una manzana podrida, pero el jefe del equipo no hace nada. El resultado es predecible: rehacer el trabajo de la manzana podrida.
- Hazañas. Algunos desarrolladores de software ponen un gran énfasis en la realización de hazañas en los proyectos. Pero lo que hacen tiene más de malo que de bueno. Ejemplo, los directivos de nivel medio dan mayores aplausos a actitudes del tipo ser capaz de que a los progresos firmes y consistentes y a los informes significativos de progreso. El resultado es un modelo de planificación al límite en el que las amenazas de desajuste del plan no se detectan, no se conocen o ni se informan a la cadena de directivos hasta el último minuto. Un pequeño equipo de desarrollo y sus jefes inmediatos toman como rehenes a una compañía entera por no admitir que tiene problemas para cumplir

su plan. El énfasis en los comportamientos heroicos fomenta correr un riesgo extremo, e impide la cooperación entre los múltiples elementos que contribuyen al proceso de desarrollo del software. Algunos directivos fomentan el comportamiento heroico cuando se concentran con demasiada firmeza en actitudes del tipo "ser capaz de". Elevando estas actitudes por encima de informes del estado exacto y a veces pesimistas, los directivos de estos proyectos coartan su capacidad de tomar medidas correctivas. Ni siquiera saben que tienen que emprender acciones correctoras hasta que el daño ya está hecho.

- Añadir más personal a un proyecto retrasado. Este es quizás el más clásico de los errores clásicos. Cuando un proyecto se alarga, añadir más gente puede quitar más productividad a los miembros del equipo existente de la que añaden los nuevos miembros.
- Oficinas repletas y ruidosas. La mayoría de los desarrolladores consideran sus condiciones de trabajo como insatisfactorias. Alrededor del 60 por 100 indican que no tienen suficiente silencio ni privacidad. Los trabajadores que están en oficinas silenciosas y privadas tienden a funcionar significativamente mejor que aquellos que ocupan cubículos en salas ruidosas y repletas. Los entornos repletos y ruidosos alargan los planes de desarrollo.
- Fricciones entre los clientes y los desarrolladores. Las fricciones entre los clientes y los desarrolladores pueden presentarse de

distintas formas. A los clientes pueden parecerles que los desarrolladores no cooperan cuando rehúsan comprometerse con el plan de desarrollo que desean los clientes o cuando fallan al entregar lo prometido. A los desarrolladores puede parecerles que los clientes no son razonables porque insisten en planes irreales o cambios en los requisitos después de que éstos hayan sido fijados. Pueden ser simplemente conflictos de personalidad entre dos grupos. El principal efecto de esta fricción es la mala comunicación, y los efectos secundarios de la mala comunicación incluyen el pobre entendimiento de los requisitos, pobre diseño de la interfaz de usuario y, en el peor caso, el rechazo del cliente a aceptar el producto acabado. En el caso medio, las fricciones entre clientes y desarrolladores de software llegan a ser tan severas que ambas partes consideran la cancelación del proyecto. Para remediar estas fricciones se consume tiempo, y distraen tanto a desarrolladores como a clientes del trabajo real en el proyecto.

- Expectativas pocos realistas. Una de las causas más comunes de fricciones entre los desarrolladores y sus clientes o los directivos son las expectativas poco realistas. Ejemplo, no tener razones técnicas para pensar que un software se podrá desarrollar en 6 meses, pero ése es el plazo en que lo quiere el comité ejecutivo de la compañía. La incapacidad del jefe de proyecto para corregir esta expectativa irreal será la principal fuente de problemas. En otros casos, los directivos o los desarrolladores de un proyecto se

buscan problemas al pedir fondos basándose en estimaciones de planificación demasiado optimistas. Aunque por sí mismas las expectativas irreales no alargan el plan, contribuyen a la percepción de que el plan de desarrollo es demasiado largo, y de que puede ser malo. Una inspección de Standish Group marcó las expectativas realistas como uno de los cinco factores principales necesarios para asegurar el éxito de los proyectos internos de software de gestión.

- Falta de un promotor efectivo del proyecto. Para soportar muchos de los aspectos del desarrollo rápido es necesario un promotor del proyecto de alto nivel, incluyendo una planificación realista, el control de cambios y la introducción de nuevos métodos de desarrollo. Sin un promotor ejecutivo efectivo, el resto del personal de alto nivel de la empresa puede forzar a que se acepten fechas de entrega irreales o hacer cambios que debiliten el proyecto. El consultor australiano Rob Thomstt afirma que la falta de un promotor efectivo garantiza virtualmente el fracaso del proyecto.
- Falta de participación de los implicados. Todos los principales participantes del esfuerzo de desarrollo de software deben implicarse en el proyecto. Incluyendo a los promotores, ejecutivos, responsables del equipo, miembros del equipo, personal de ventas, usuarios finales, clientes y cualquiera que se juegue algo con el proyecto. La cooperación estrecha sólo se produce si se han implicado todos los participantes, permitiendo

una coordinación precisa del esfuerzo para el desarrollo rápido, que es imposible conseguir sin una buena participación.

- Falta de participación del usuario. La inspección de Standish Group descubrió que la razón número uno de que los proyectos de Sistemas de Información tuviesen éxito es la implicación del usuario. Los proyectos que no implican al usuario desde el principio corren el riesgo de que no se comprendan los requisitos del proyecto, y son vulnerables a que se consuma tiempo en prestaciones que más tarde retrasarán el proyecto.
- Política antes que desarrollo. Larry Constantine indicó que si hay cuatro tipos diferentes de orientaciones políticas. Los "políticos" están especializados en la "gestión", centrándose en las relaciones con sus directivos. Los "investigadores" se centran en explorar y reunir la información. Los "aislacionistas" están solos, creando fronteras para el proyecto que mantienen cerradas a los que no son miembros del equipo. Los "generalistas" hacen un poco de todo: establecen relaciones con sus directivos, realizan investigaciones y exploran actividades, y se coordinan con otros equipos como parte de su modo de trabajo. Constantine indicó que inicialmente los equipos políticos y generalistas están bien vistos por los directivos de alto nivel. Pero después de un año y medio, los equipos políticos llegan a la muerte súbita. Primar la política en vez de los resultados es fatal para el desarrollo orientado a la velocidad.

- Ilusiones. Muchos problemas del desarrollo del software se deben a la ilusión. Cuántas veces hemos escuchado cosas como éstas a distintas personas: "Ninguno de los miembros del proyecto cree realmente que pueda completarse el proyecto de acuerdo con el plan que tienen, pero piensan que quizás si trabajan duro, y nada va mal, y tienen un poco de suerte, serán capaces de concluir con éxito". "Nuestro equipo no hace mucho trabajo para la coordinación de las interfaces entre las distintas partes del producto, pero tenemos una buena comunicación para otras cosas, y las interfaces son relativamente simples, así que probablemente sólo necesitaremos un día o dos para eliminar los errores". "Sabemos que contamos con un desarrollador externo de poco talento para el subsistema de la base de datos, y que es difícil ver cómo va a acabar el trabajo con los niveles de personal que ha especificado en su propuesta. No tienen tanta experiencia como algunos de los demás desarrolladores externos, pero puede que compensen con energía lo que les falta en experiencia. Probablemente acaben a tiempo". "No necesitamos reflejar la última lista de cambios en el prototipo para el cliente. Estoy seguro de que por ahora sabemos lo que quiere". "El equipo está diciendo que realizará un esfuerzo extraordinario para cumplir con la fecha de entrega, y que no han llegado a su primer hito por pocos días, pero creo que alcanzarán éste a tiempo". Las ilusiones no son sólo optimismo. Realmente consisten en cerrar los ojos y esperar que todo funcione cuando no se tienen las

bases razonables para pensar que será así. Las ilusiones al comienzo del proyecto llevan a grandes explosiones al final e impiden llevar a cabo una planificación coherente y pueden ser la raíz de más problemas en el software que todas las otras causas combinadas.

2.2.1.7 Proceso.

Los errores relacionados con el proceso malgastan el talento y el esfuerzo del personal. A continuación se muestran algunos de los peores errores relacionados con el proceso (Pressmann, 2005) (Sommerville, 2005):

- Planificación excesivamente optimista. Los retos a los que se enfrenta alguien que desarrolla una aplicación en tres meses son muy diferentes de aquellos a los que se enfrenta alguien que desarrolla una aplicación que necesita un año. Fijar un plan excesivamente optimista predispone a que el proyecto falle por infravalorar el alcance del proyecto, minando la planificación efectiva, y reduciendo las actividades críticas para el desarrollo, como el análisis de requisitos o el diseño. También supone una excesiva presión para los desarrolladores, quienes a largo plazo se ven afectados en su moral y su productividad.
- Gestión de riesgos insuficiente. Algunos errores no son lo suficientemente habituales como para considerarlos clásicos. Son los llamados "riesgos". Como con los errores clásicos, si no ejercemos una gestión activa de los riesgos, con qué sólo vaya mal una cosa se pasará de tener un proyecto con un desarrollo rápido a uno con un desarrollo

lento. El fallo de no gestionar uno solo de estos riesgos es un error clásico.

- Fallos de los contratistas. Las compañías a veces contratan la realización de partes de un proyecto cuando tienen demasiada prisa para hacer el trabajo en casa. Pero los contratados frecuentemente entregan su trabajo tarde, con una calidad inaceptable o que falla al no coincidir con las especificaciones. Riesgos como requisitos inestables o interfaces mal definidas pueden ser enormes cuando un contratado entre en escena. Si las relaciones con los contratados no se gestionan cuidadosamente, la utilización de desarrolladores externos pueden retardar el proyecto en vez de acelerarlo.
- Planificación insuficiente. Si no planificamos para conseguir un desarrollo rápido, no podemos esperar obtenerlo.
- Abandono de planificación bajo presión. Los equipos de desarrollo hacen planes y rutinariamente los abandonan cuando se tropiezan con un problema en la planificación. El problema no está en el abandono del plan, sino más bien en fallar al no crear un plan alternativo, y caer entonces en el modo de trabajo de codificar y corregir. Ejemplo, un equipo abandona su plan después de fallar en la primera entrega, y esto es lo habitual. A partir de este punto, el trabajo no tiene coordinación ni elegancia.
- Pérdida de tiempo en el inicio difuso. El "inicio difuso" es el tiempo que transcurre antes de que comience el proyecto; este tiempo normalmente se pierde en el proceso de aprobar y hacer el presupuesto. No es poco común que un proyecto desperdicie meses o años en un inicio difuso, y

entonces se está a las puertas de un plan agresivo. Es mucho más fácil y barato y menos arriesgado suprimir unas pocas semanas o meses del inicio difuso en vez de comprimir el plan de desarrollo en ese mismo tiempo.

- Escatimar en las actividades iniciales. Los proyectos se aceleran intentando acortar las actividades "no esenciales", y puesto que el análisis de requisitos, la arquitectura y el diseño no producen código directamente, son los candidatos fáciles. Los resultados de este error, también conocido como "saltar a la codificación", son todos demasiado predecibles. Los proyectos que normalmente escatiman en sus actividades iniciales tendrán que hacer ese trabajo en otro momento, con un costo de 10 a 100 veces superior a haberlo hecho bien inicialmente. Si no podemos encontrar cinco horas para hacer el trabajo correctamente la primera vez, ¿cómo vamos a encontrar 50 para hacerlo correctamente más tarde?
- Diseño inadecuado. Un caso especial de escatimar en las actividades iniciales es el diseño inadecuado. Proyectos acelerados generan un diseño indeterminado, no asignado suficiente tiempo para él y originado un entorno de alta presión que hace difícil la posibilidad de considerar alternativas en el diseño. El énfasis en el diseño está más orientado a la conveniencia que a la calidad, por lo que necesitará varios ciclos de diseño de poder finalizar completamente el sistema.
- Escatimar en el control de calidad. En los proyectos que se hacen con prisa se suele cortar por lo sano, eliminando las revisiones del diseño y del código, eliminando la planificación de las pruebas y realizando sólo

pruebas superficiales. Acortar en un día las actividades de control de calidad al comienzo del proyecto probablemente supondrá de 3 a 10 días de actividades finales.

- Control insuficiente de la directiva. Poco control de la directiva para detectar a tiempo los signos de posibles retrasos en el plan, y los pocos controles definidos al comienzo se abandonan cuando el proyecto comienza a tener problemas. Antes de encarrilar un proyecto, en primer lugar debemos ser capaces de decir si va por buen camino.
- Convergencia prematura o excesivamente frecuente. Bastante antes de que se haya programado entregar un producto, hay un impulso para preparar el producto para la entrega, mejorar el rendimiento del producto, imprimir la documentación final, incorporar entradas en el sistema final de ayuda, pulir el programa de instalación, eliminar las funciones que no van a estar listas a tiempo y demás. En proyectos hechos con prisa, hay una tendencia a forzar prematuramente la convergencia. Puesto que no es posible forzar la convergencia del producto cuando se desea, algunos proyectos de desarrollo rápido intentan forzar la convergencia media docena de veces o más antes de que finalmente se produzca. Los intentos adicionales de convergencia no benefician al producto, Sólo son una pérdida de tiempo y prolongan el plan.
- Omitir tareas necesarias en la estimación. Si la gente no guarda cuidadosamente datos de proyectos anteriores, olvida las tareas menos visibles, pero son tareas que se han de añadir. El esfuerzo omitido suele aumentar el plan de desarrollo en un 20 o 30 por 100.

- Planificar ponerse al día más adelante. Un tipo de reestimación es responder inapropiadamente el retraso del plan. Si hemos trabajado en un proyecto durante 6 meses, y hemos empleado tres meses en llegar al hito correspondiente a los dos meses ¿qué hacer?. En muchos proyectos simplemente se plantea recuperar el retraso más tarde, pero nunca se hace. Aprenderemos más del producto conforme lo estamos construyendo, incluyendo más sobre lo que nos llevará construirlo. Estos conocimientos necesitan reflejarse en la reestimación del plan. Otro tipo de error es la reestimación que se debe a cambios en el producto. Si el producto que estamos construyendo cambia, la cantidad de tiempo necesaria para construirlo cambiará también. El crecimiento de las nuevas prestaciones sin ajustar el plan garantiza que no se alcanzará la fecha de entrega.
- Programación a destajo. Algunas organizaciones creen que la codificación rápida, libre, tal como salga, es el camino hacia el desarrollo rápido. Piensan que si los desarrolladores están lo suficientemente motivados, pueden solventar cualquier obstáculo. Este enfoque muchas veces se presenta como un enfoque "empresarial" al desarrollo de software, pero realmente es sólo la envoltura del viejo paradigma a destajo combinado con una planificación ambiciosa, y esta combinación raras veces funciona. Es un ejemplo de que dos negaciones no constituyen una verdad.

2.2.2 Pruebas de software

Existen muchas formas de organizar desarrollo de las pruebas de software, desde estructuras sencillas en las que es el propio programador el que las realiza hasta organizaciones que proporcionan servicios orientados al desarrollo de las actividades de verificación y validación de productos software.

La decisión sobre la estructura que mejor se adapta a cada organización depende de la política y cultura de la misma así como del conocimiento y habilidades de los recursos humanos implicados en el proceso de pruebas software con que cuenten. Se presenta un breve análisis de las estructuras organizativas más relevantes que son implementadas en las organizaciones software, puesto que la estructura organizativa es uno de los condicionantes a la hora de implementar procesos de pruebas de software. Además, es necesario conocerlas para poder determinar qué modelo de referencia se ajusta mejor a cada una de ellas.

2.2.2.1 Los Desarrolladores son Testers.

En las organizaciones que aplican este tipo de estructura, los desarrolladores del producto software son también los encargados de realizar todas las actividades relacionadas con las pruebas software (Koirala y Sheikh, 2008). La principal ventaja de esta estructura es que los desarrolladores comprenden completamente el producto que están probando y tienen pleno conocimiento del diseño y código. Sin embargo, ésta constituye también uno de los principales inconvenientes, este conocimiento provoca que, de forma inconsciente, las pruebas se diseñen

para ser pasadas sin dar lugar a errores y no para detectar defectos (Craig y Jaskiel, 2002). El tiempo es una variable importante dentro de esta estructura, dado que el desarrollador es el encargado de realizar las pruebas cuando se sienta presionado por el tiempo de desarrollo, utilizará el tiempo de pruebas para el desarrollo y disminuirá la carga de estas últimas. Además, los desarrolladores disponen del conocimiento y habilidades necesarias para realizar el desarrollo de un producto pero no tienen las competencias requeridas para realizar las actividades de pruebas (Kit, 1995).

2.2.2.2 Equipos de prueba integrados

En este tipo de estructura las organizaciones tratan de integrar al equipo de pruebas en el equipo de desarrollo a través del jefe de proyecto. La principal ventaja es que los integrantes del equipo poseen los conocimientos y habilidades necesarias para desempeñar sus tareas correctamente, y además se encuentran integrados en el equipo de desarrollo (Hetzel, 1988) (Cohen et al., 2004). Por tanto, la comunicación entre ambos es más fluida y las relaciones menos tensas, comparten recursos y trabajan como un equipo. Aunque conseguir que trabajen como un único equipo es una tarea muy compleja que no siempre se desarrolla con éxito, el principal inconveniente se produce cuando el equipo comienza a estar bajo presión, la actitud del jefe de proyecto habitualmente apoya a terminar el producto sacrificando la calidad del mismo, es decir disminuyendo las actividades de pruebas (Craig y Jaskiel, 2002).

2.2.2.3 Equipos de pruebas independientes

En este tipo de estructura las organizaciones definen un equipo de pruebas, que es independiente del equipo de desarrollo, cuyo trabajo es realizar todas las actividades relacionadas con las pruebas software (Black, 2004). La principal ventaja es que los integrantes del equipo poseen los conocimientos y habilidades necesarias para desempeñar sus tareas correctamente (Burnstein, 2003); es decir, el equipo se compondrá por personal experimentado en el área de pruebas. Sin embargo, la creación de este equipo frecuentemente constituirá la creación de un muro entre desarrolladores y testers, lo que dificulta en gran medida la comunicación necesaria entre ambos equipos (Craig y Jaskiel, 2002). El equipo de pruebas carece de conocimiento sobre el producto por lo que necesita que éste sea transferido por el equipo de desarrollo, si la comunicación entre ambos equipos no es fluida éste constituirá un problema muy importante.

2.2.2.4 Grupos de SQA

En este tipo de estructura las organizaciones asignan al personal de SQA las funciones de verificación y validación. Esto se debe a que las habilidades de los integrantes del grupo de SQA son similares a las habilidades que poseen los testers. La principal ventaja es que estos grupos trabajan en la organización y tienen habilidades para desempañar las actividades de pruebas software (Craig y Jaskiel, 2002). Además, están mejor vistos por los desarrolladores ya que no se trata del equipo de

pruebas sino del grupo de aseguramiento de la calidad (Black, 2004). El principal inconveniente es que el grupo de SQA no sólo realiza pruebas, sino que tiene otras tareas que desempeñar. Estas responsabilidades adicionales provocan que no puedan desempeñar su trabajo de forma efectiva (Hetzel, 1988).

2.2.2.5 Factoría de Pruebas

Este tipo de estructura está basado en el concepto de outsourcing. (McCarthy y Anagnostou, 2004) definen outsourcing como: "el acuerdo comercial mediante el cual una organización externaliza una parte de su actual negocio a otra organización". La evolución del trabajo y los procesos de negocio están propiciando que cada vez más las organizaciones subcontraten servicios y desarrollos a otras organizaciones (Smite, 2006) (Ejaz, 2006). Hasta el momento la tendencia ha sido externalizar el trabajo más simple y rutinario a trabajadores menos cualificados y a un menor coste (Donahoe y Pecht, 2003); sin embargo, en la actualidad se empieza a hacer outsourcing de actividades que necesitan de habilidades e infraestructura específica, como es el caso de las pruebas software (Simon et al., 2009). El objetivo principal que se persigue con la externalización de determinados servicios es conseguir un incremento de la eficiencia, una reducción de los costes y fomentar la innovación.

La principal ventaja de este enfoque es que las organizaciones especializadas disponen del personal experimentado y la infraestructura

necesaria para llevar a cabo de una forma controlada y adecuada las diferentes actividades relacionadas con las pruebas software (Weigelt, 2009). Por el contrario, los inconvenientes se basan en el desconocimiento de la visión funcional de los aspectos de negocio de la organización contratante (Craig y Jaskiel, 2002).

Diferentes autores, entre ellos (Taipale et al., 2006), (Kaner et al., 1999) soportan que el outsourcing de las pruebas software constituyen una buena forma de probar los productos con un elevado nivel de calidad. Sin embargo, ésta estructura no está libre de inconvenientes, aunque ya se ha citado alguno, la principal desventaja es la necesidad de realizar una buena gestión de las actividades externalizadas (Craig y Jaskiel, 2002), ésta constituirá el principal factor clave para el éxito del outsourcing de pruebas software.

Además, en este tipo de entornos aparece también otro riesgo importante que es la excesiva dependencia en el proveedor que puede llevar a la pérdida de control sobre las actividades de prueba o la pérdida de "know-how" sobre la ejecución de dichas actividades (González et al., 2005). En relación a los costes que tiene la externalización de actividades (Barthelemy, 2001) establece que existen una serie de costes ocultos como los relativos a la transición o a la gestión del proveedor que no son siempre considerados por las organizaciones subcontratadoras.

Cómo se hacía mención, en la actualidad existen diferentes modelos de referencia que han sido definidos con el propósito de mejorar el proceso de pruebas software y, por tanto, la calidad de los productos. Si bien la aplicabilidad de dichos modelos está condicionada por el tipo de estructura organizativa que implemente la organización. Posteriormente se analizan estos modelos y se destacan cuáles de ellos son apropiados y cuáles no para ser implantados en cada una de las estructuras organizativas presentadas en este epígrafe.

Sin embargo, la estructura organizativa que implementa una organización puede ser modificada como consecuencia de decisiones estratégicas o presupuestarias, mientras que el modelo de procesos a seguir no debería de cambiar puesto que éste debe de formar parte de la cultura de la organización. La solución propuesta es aplicable desde la estructura más simple como puede ser que el equipo de desarrollo se encargue de realizar el proceso de pruebas, hasta la más compleja que sería la externalización de estas actividades a Factorías de Pruebas Software. Esto se debe a que los procesos han sido diseñados y definidos de forma modular, permitiendo que cada organización implemente únicamente aquellos que se ajusten a sus necesidades.

2.2.2.6 Principales modelos de referencia de buenas prácticas

Existen diferentes modelos de referencia que definen el conjunto de buenas prácticas a realizar dentro de una organización con el objetivo de llevar a cabo una mejora de procesos. Es decir, estos modelos

proporcionan el marco de referencia necesario para determinar el conjunto de fortalezas y debilidades de los procesos implementados en una organización y, determinar, a partir de las mismas, las acciones de mejora a realizar. CMMI es el modelo de referencia más difundido dentro de la industria del software, proporciona la cobertura necesaria para el desarrollo de actividades aplicadas tanto a productos como a servicios.

Además, existen modelos de referencia dirigidos especialmente al proceso de pruebas de software, los más destacables son: TMM (Burnstein 2003), TMMi (Veenendaal 2010) y TPI (Koomen 1999). Estos modelos ofrecen el marco de referencia necesario para determinar el conjunto de fortalezas y debilidades en el área de pruebas de software. Además, se incluye el análisis de TMap - Test Management approach (Koomen, 1999) ya que, aunque no se trata de un modelo de referencia propiamente dicho, es una metodología, basada en la experiencia y conocimiento de la organización que lo ha creado, para gestionar y realizar las pruebas de software de una forma estructurada que se encuentra bastante extendida.

2.2.2.6.1CMMI for Development

CMMI for Development (Capability Maturity Model Integration for Development) es un modelo de referencia que cubre el desarrollo y mantenimiento de las actividades aplicadas tanto a los productos como a los servicios. CMMI surge a partir de la versión 2.0 de SW-CMM con objetivo de solucionar los

problemas existentes con los distintos CMM. La misión inicial de este proyecto fue combinar los modelos SW-CMM, SECM e IPD-CMM. Su primera versión fue publicada en agosto del año 2000. Poseía un núcleo común a los diferentes modelos CMM y después una parte específica propia de cada modelo; dos años después, se publicó la versión 1.1.

Sin embargo, existía la necesidad de integrar a las diferentes disciplinas en un marco de trabajo combinado que permitiese dirigir tanto Software como Sistemas de Ingeniería o ambos, junto con IPPD. En Noviembre de 2010 se publica CMMI for Development v1.3, que es la versión más actual en lo que se refiere a buenas prácticas para el desarrollo de productos software. Ésta, además de ser una versión integrada al igual que la 1.1 y la 1.2, proporciona mejoras sobre la versión anterior, para solucionar los problemas que han ido surgiendo de la práctica. Es importante también reseñar que esta última versión del CMMI. Consta de otras dos extensiones, llamadas constelaciones, dirigidas a la mejora de los procesos de adquisición de productos y servicios (CMMI for Acquisitions) y a la mejora de los procesos de gestión del servicio (CMMI for services). Estos dos últimos no han sido estudiados en esta tesis doctoral, puesto que no incluyen buenas prácticas referentes a la validación y verificación de productos software.

CMMI define un conjunto de buenas prácticas que cubren un conjunto de áreas de proceso involucradas en el desarrollo y mantenimiento, permitiendo establecer tanto el nivel de madurez como el nivel de capacidad existente en la organización.

Permite dos representaciones diferentes: continua y por etapas. La primera define 6 niveles de capacidad que permiten conocer la capacidad de los procesos que sean seleccionados para su mejora; se centra en un área de proceso y en adquirir un nivel de capacidad para el mismo. La segunda define cinco niveles de madurez: Inicial, Gestionado, Definido, Gestionado Cuantitativamente y Optimizado, que permiten conocer la madurez de los procesos que se realizan en la organización; centrándose en la madurez de toda la organización.

En relación al proceso de pruebas define 3 áreas de proceso: Product Integration, Validation, y Verification; sin embargo no cubren todas las necesidades del proceso de pruebas, puesto que no se encuentran definidas todas las mejores prácticas relacionadas con las pruebas software ni el conjunto de los productos de trabajo que son necesarios. Esto se debe a que, al igual que el resto de las áreas de proceso de CMMI, este estándar no entra al detalle de cómo deben de definirse e implementarse los procesos. El propósito del área de Product Integration es llevar a cabo la integración de los distintos componentes que forman el

producto, incluyendo las actividades necesarias para asegurar la compatibilidad de las distintas interfaces.

En relación al área de Validation, su propósito es demostrar que un producto o un componente de un producto satisfacen las necesidades para las cuales ha sido creado, dentro del entorno en el que deberá ser ejecutado. Para ello, CMMI propone dos actividades a llevar a cabo: Preparación para la validación y Validación de los productos o componentes.

Respecto al área de Verification, su propósito es asegurar que un producto cumple con los requisitos que hayan sido especificados. Incluye la verificación tanto del producto final como de un producto de trabajo intermedio con respecto a los requisitos. Es un proceso incremental que ocurre a través del desarrollo del producto y productos de trabajo, comienza con la verificación de los requisitos, continua con la verificación de la evolución de los productos de trabajo, y finaliza con la verificación del producto completo. Para ello, CMMI propone llevar a cabo tres actividades: Preparación para la verificación, Desarrollo de revisiones por pares y Verificación de los productos de trabajo seleccionados.

Como se ha visto, CMMI for Development 1.3 es un estándar general para la mejora de los procesos de desarrollo de software,

que define un conjunto de niveles y áreas de proceso a ser implementadas por las organizaciones para mejorar el desarrollo de productos software, considerando todas las fases que aparecen en el proceso. Sin embargo, este estándar es genérico y no define en detalle procesos ni prácticas a ser implementadas en ninguna de sus áreas de proceso, por lo que requiere del uso de otros estándares y de la experiencia del personal para definir los procesos adecuados dentro de la organización.

Además, dada su generalidad, no cubre el proceso de pruebas en toda su amplitud, simplemente define tres áreas de proceso que, como se ha visto, sólo incluyen algunas de las actividades básicas relacionadas con las pruebas. Por ello, este estándar no puede ser utilizado como modelo para la mejora del proceso de prueba en las organizaciones.

2.2.2.6.2 TMM - Test Maturity Model

El modelo TMM-Test Maturity Model (Burnstein 2003) ha sido desarrollado por el Instituto Tecnológico de Illinois como una guía para la mejora del proceso de pruebas. Al igual que CMM, modelo al que complementa, utiliza el concepto de niveles de madurez para la evaluación y mejora del proceso de pruebas; aunque a diferencia de su antecesor no contempla los niveles de capacidad.

TMM consta de 5 niveles de madurez que reflejan el grado de madurez de la organización en el proceso de pruebas; es decir, representan la evolución hacia un proceso de pruebas maduro. Cada nivel tiene definidas un conjunto de áreas de proceso, y cada una de ellas contiene las actividades relacionadas con la misma que es necesario realizar para contribuir a la mejora del proceso (Burnstein, 2003).

De acuerdo a (Burnstein, 2003), cada uno de los niveles tiene una serie de características. El nivel inicial se caracteriza porque el proceso de pruebas es un caos; no se encuentra definido y es considerado como una parte de la depuración. Las pruebas se desarrollan ad hoc después de haber terminado toda la codificación. En el nivel 2, se separan las pruebas de la depuración, definiéndose una fase después de la codificación. El objetivo principal de este nivel es verificar que el software satisface los requisitos especificados. Sin embargo, surgirán problemas de calidad en este nivel, debido a que la planificación de las pruebas se realiza tarde. Además, los defectos se propagarán desde los requisitos y fases de diseño al código, ya que no existen programas de revisión.

En el nivel 3, las pruebas se encuentran integradas en todo el ciclo de vida del software, no son una fase que se realiza después de la codificación. A diferencia del nivel 2, la planificación de las

pruebas comenzará con la fase de requisitos y continuará durante todo el ciclo de vida. Existe una organización de pruebas, un programa de formación y se reconoce a las pruebas como una actividad profesional. Se realizarán revisiones aunque no exista un programa formal de revisión y éstas resulten inconsistentes.

El nivel 4 se caracteriza porque las pruebas son un proceso medido y cuantificado. Las revisiones de todas las fases del proceso de desarrollo son reconocidas como actividades de pruebas y control de calidad en este nivel. Se prueba el software atendiendo a los atributos de calidad, como fiabilidad, usabilidad y mantenimiento. Las deficiencias en este nivel se producen debido a la carencia en la prevención de defectos. Las actividades de prueba se expanden, incluyen revisiones, inspecciones y walkthroughs a través de las diferentes fases.

Finalmente, el nivel 5 se caracteriza porque el proceso de pruebas es repetible, definido, gestionado y medido, por lo que se pueden definir mecanismos que permitan una mejora continua en el proceso de pruebas. Se realiza prevención de defectos y control de calidad.

TMM podría ser utilizado en organizaciones que utilizan una estructura organizativa de pruebas basada en equipos independientes o grupos de SQA. En el caso de que los

desarrolladores sean los testers o equipos integrados de pruebas este modelo de referencia resulta demasiado complejo y burocrático como para ser utilizado; tampoco es apropiado para una factoría de pruebas ya que no contempla procesos de gestión de servicios.

2.2.2.6.3 TMMi - Test Maturity Model Integration

El modelo TMMi - Test Maturity Model Integration (Veenendaal 2010) ha sido desarrollado por la TMMi Foundation como una guía y un marco de referencia para la mejora del proceso de pruebas, siendo un modelo complementario a CMMI. Al igual que este último, TMMi define una representación por etapas y hace uso del concepto de niveles de madurez para evaluar y mejorar las diferentes áreas de proceso de cada uno de los niveles.

Además de las áreas de proceso identifica el conjunto de objetivos a alcanzar y las prácticas que hay que realizar para ello. El objetivo de TMMi es soportar las actividades de prueba y la mejora del proceso de prueba tanto en las disciplinas de ingeniería de sistemas y de ingeniería de software. Al igual que CMMI, TMMi trata de proporcionar a las organizaciones la mejora de procesos, en este caso de procesos de prueba, para lo cual es sustancial contar con un mecanismo de evaluación que permita identificar las oportunidades de mejora. Para ello la TMMi Foundation ha elaborado el "TMMi Assessment Method

Application Requirements", que contiene los requisitos para la evaluación de TMMi, de forma que se puede conocer el nivel de madurez de una organización en lo referente al proceso de pruebas (Goslin 2009).

TMMi consta de cinco niveles que van desde la realización de un proceso de pruebas de manera ad-hoc y no gestionada hasta optimizado pasando por gestionado, definido, y medido optimizado. Pasar de un nivel a otro supone haber alcanzado una mejora adecuado y sirve también como punto de partida para el siguiente nivel (Veenendaal 2010).

El nivel 1 se caracteriza porque las pruebas se realizan de manera caótica, no hay un proceso definido y, frecuentemente, sólo son consideradas como parte de la depuración. Las pruebas se realizan de manera ad-hoc una vez que la codificación se ha terminado, puesto que el objetivo de las pruebas a este nivel es comprobar si el software se ejecuta sin errores importantes. El éxito depende fundamentalmente de la competencia del personal a cargo de las mismas (Veenendaal 2010).

En el nivel 2 se dispone de un proceso gestionado para la realización de las pruebas que está separado claramente de la depuración. El principal objetivo de este nivel es verificar que el producto satisface los requisitos establecidos inicialmente. Consta

de cinco áreas de proceso que son: Política y estrategia de pruebas, Planificación de pruebas, Monitorización y control de pruebas, Diseño y ejecución de pruebas y Entorno de prueba (Veenendaal 2010).

A nivel 3 el proceso de pruebas se encuentra integrado dentro del ciclo de vida de desarrollo y cuenta con sus propios hitos. Las organizaciones que se encuentran en este nivel entienden la importancia de las revisiones y del control de calidad. Las áreas de proceso definidas para este nivel son: Organización de pruebas, Programa de formación en pruebas, Ciclo de vida e integración de pruebas, Pruebas no funcionales y revisiones por pares (Veenendaal 2010).

Las organizaciones que se encuentran en el nivel 4 de TMMi disponen de un proceso de pruebas bien fundado, completamente definido y medible. Las pruebas se perciben como evaluación, por lo que las organizaciones ponen en práctica programas de mejorar del proceso de pruebas evaluación su calidad, la productividad y haciendo seguimiento de las mejoras. Este nivel consta de tres áreas de proceso: Medición de pruebas, Evaluación de la calidad del producto y revisión por pares avanzada (Veenendaal 2010).

Finalmente, el nivel 5 se caracteriza porque la organización es capaz de mejorar continuamente su proceso de pruebas en base a

un proceso controlado estadísticamente. Los métodos y técnicas de pruebas son optimizados y existe un foco continuo en la mejora del proceso. Para alcanzar este objetivo se dispone de tres áreas de proceso: Prevención de defectos, Control de Calidad y Optimización del proceso de prueba (Veenendaal 2010).

Este modelo, al igual que CMMI, tiene una gran desventaja a la hora de ser empelado por las organizaciones, puesto que no contiene una definición clara de los procesos, actividades y tareas a ejecutar, elementos de trabajo, productos de trabajo, etc. Por tanto, no existe una descripción formal del proceso; si no que simplemente proporciona información a ser considerada en la definición de los mismos a alto nivel. Este hecho frena su utilización en las organizaciones, ya que no proporciona la guía necesaria para la propia definición de los procesos de prueba (Steiner et al. 2010).

2.2.2.6.4 TPI - Test Process Improvement

TPI -Test Process Improvement (Koomen 1999) fue definido para una mejora controlada y gradual del proceso de pruebas de software, basándose en el conocimiento y la experiencia de sus autores, consultores de pruebas de Sogeti. TPI define un conjunto de 20 áreas clave, que son: Estrategia de pruebas, Modelo del ciclo de vida, Momento de implicación, Estimación y planificación, Técnicas de especificación de pruebas, Técnicas de

pruebas estáticas, Métricas, Automatización de pruebas, Entorno de pruebas, Entorno de trabajo, Compromiso y motivación, Funciones de prueba y entrenamiento, Alcance de la metodología, Comunicación, Informes, Gestión de defectos, Gestión de elementos de prueba, Gestión del proceso de pruebas, Evaluación y Pruebas de bajo nivel (Koomen 1999).

Cada una de las áreas clave puede tener un determinado nivel de capacidad (ascendiendo desde la A hasta la D). El ascenso de nivel implica una mejora en tiempo, dinero y/o calidad. Los requisitos que permiten pasar de un nivel a otro son definidos en forma de preguntas que deben ser respondidas positivamente para alcanzar cierto nivel, siendo necesario haber logrado el nivel anterior para pasar al siguiente; es decir no se podrá estar en un nivel B si no se cumplen los requisitos del nivel A. A este conjunto de preguntas se le denominará puntos de validación, ya que permiten determinar el nivel en el que se encuentra el proceso para cada área clave (Koomen 1999).

Además, como no todas las áreas y niveles son igual de importantes y existen dependencias entre ellas, TPI define una matriz de madurez de pruebas que establece las diferentes relaciones (Koomen 1999).

La primera limitación que surge a la hora de utilizar TPI es que está estrechamente ligado a la metodología TMap por lo que sería necesario que la organización implementara dicha metodología como mecanismo para la gestión del proceso de pruebas. El uso de esta metodología tiene también una serie de desventajas que hacen que sea difícil de implantar en las organizaciones, especialmente cuando esta son pequeñas y medianas empresas (Pymes).

Además, tal y como se ha mencionado, TPI debe ser considerado como una herramienta para estructurar las acciones de mejora del proceso de pruebas y como un medio para la comunicación, pero no es un mecanismo que contribuya a la definición de los procesos de prueba y, por tanto, no es directamente implementable en organizaciones que tratan de definir sus procesos de prueba. TPI, será únicamente utilizable de cara a mejorar los mismos en etapas posteriores. Este hecho se resalta aún más cuando se analiza su estructura, puesto que no define actividades ni tareas a ser ejecutadas para la realización del proceso de pruebas, ya que tan sólo define áreas clave y niveles que debería de tener la organización, pero a nivel de evaluación del proceso existente.

2.2.2.6.5 TMap - Test Management Approach

TMap es una aproximación para la gestión de las pruebas que ha sido desarrollado por Sogeti. TMap considera pruebas dirigidas a resultados de productos software, tanto a alto como a bajo nivel. El método proporciona respuestas a preguntas sobre pruebas, como: ¿Qué?, ¿Cuándo?, ¿Cómo?, ¿Dónde?, y ¿Quién? (Koomen et al. 2006). El contenido específico que propone TMap considera cuatro elementos fundamentales:

- Gestión de pruebas dirigida al negocio (BDTM).
- Proceso de pruebas estructurado.
- Proporcionar las herramientas adecuadas.

2.2.3 Cooperativas de Ahorro y Crédito

El texto único ordenado de la Ley General de Cooperativas, Decreto Supremo N° 074-90-TR, fue aprobado por el Presidente Alberto Fujimori Fujimori modificando los Decretos Legislativos N° 141 y 592 de la Ley General de Cooperativas. Se declara de necesidad nacional y utilidad pública, la promoción y la protección del Cooperativismo, como un sistema eficaz para contribuir al desarrollo económico, al fortalecimiento de la democracia y a la realización de la justicia social.

Asimismo, el Estado garantiza el libre desarrollo del Cooperativismo y la autonomía de las organizaciones cooperativas por lo que toda organización cooperativa debe constituirse sin propósito de lucro, y procura mediante el esfuerzo propio y la ayuda mutua de sus miembros, el servicio inmediato de éstos y el mediato de la comunidad.

La ley menciona que toda cooperativa tiene el deber de observar los siguientes principios:

- Libre adhesión y retiro voluntario;
- Control democrático;
- Limitación del interés máximo que pudiera reconocerse a las aportaciones de los socios;
- Distribución de los excedentes en función de la participación de los socios en el trabajo común o en proporción a sus operaciones con la cooperativa;
- Fomento de la educación cooperativa;
- Participación en el proceso de permanente integración;
- Irrepartibilidad de la reserva cooperativa.

Así como de cumplir con las siguientes normas básicas:

- Mantener estricta neutralidad religiosa y política partidaria;
- Reconocer la igualdad de derechos y obligaciones de todos los socios, sin discriminación alguna;
- Reconocer a todos los socios el derecho de un voto por persona, independientemente de la cuantía de sus aportaciones;
- Tener duración indefinida;
- Estar integrada por un número variable de socios y tener capital variable e ilimitado, no menores a los mínimos que, de acuerdo con su tipo o grado, le corresponda según el Reglamento.

3.1. Introducción

Para efecto del presente trabajo de investigación, se toma en cuenta del método de pruebas de software denominado CICLO-P. Este es un método de pruebas cuya fortaleza principal es lograr una correcta adherencia al proceso productivo de software y como consecuencia se obtiene una disminución en los tiempos de pruebas y por ende en los costos de las mismas. La correcta adherencia al ciclo de vida trae consigo otras implicaciones de igual importancia, entre ellas se encuentran: lograr un proceso productivo global de mayor calidad dado que los errores más críticos deben ser descubiertos en las primeras fases del ciclo, se mitigan los riesgos inherentes a un proceso productivos de software ocasionados generalmente por las estimaciones de tiempos, costos y esfuerzo, y además se fortalece el proceso de cierre de defectos y las prácticas de programación.

El método cuenta con otros pilares secundarios para garantizar la eficiencia y efectividad de las pruebas, entre ellas se encuentran: el uso clasificado de las pruebas sobre un producto permite identificar claramente que aspectos o características de este se encuentran en un estado conforme, el uso de técnicas especializadas de pruebas permite seleccionar datos de pruebas que revelen la mayor cantidad de defectos, la correcta administración de los casos de prueba permite su reutilización y por lo tanto una disminución en el tiempo de diseño y los criterios para las regresiones permiten identificar que módulos del producto de software deben probarse nuevamente a consecuencia del cierre de un defecto.

La figura 1 muestra el resumen de la técnica propuesta:

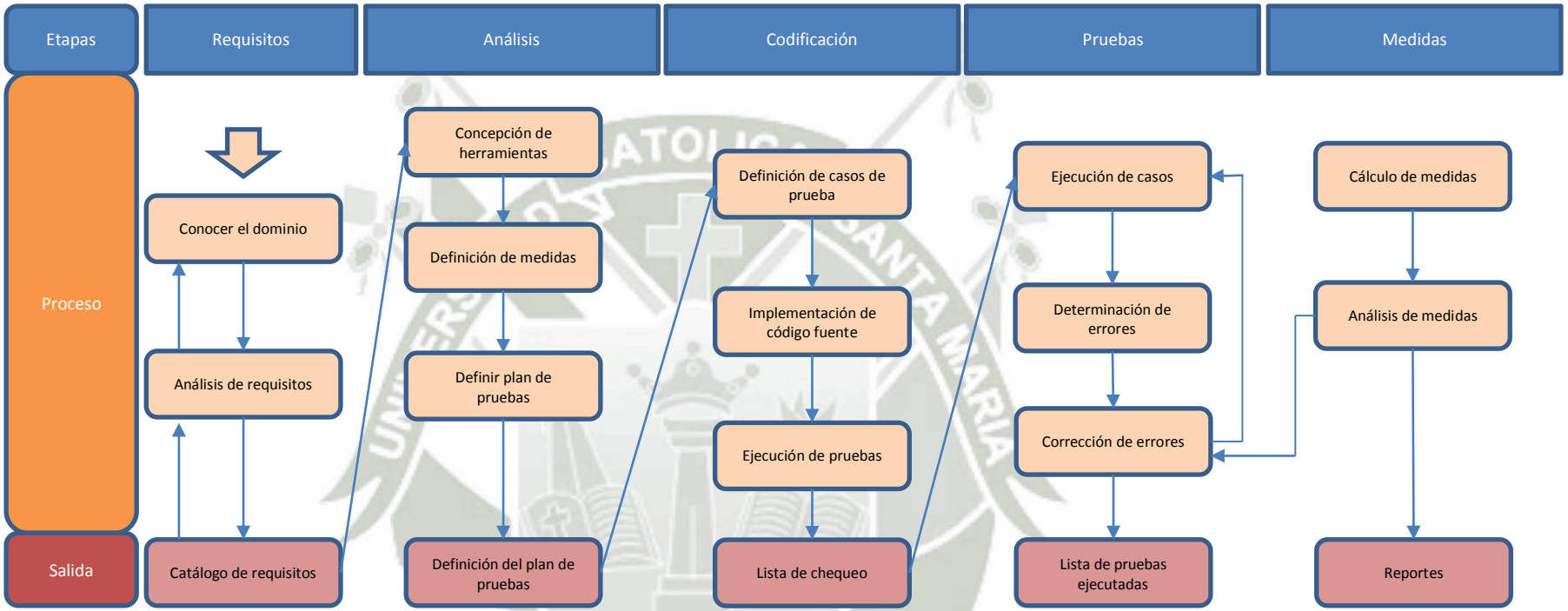


Figura Nº 1. Resumen de la técnica propuesta

3.2. Definición de condiciones de aplicación de la técnica

- La organización debe estar comprometida con los procesos de pruebas y de calidad.
- La organización debe estar en capacidad de asumir los tiempos y costos que implica la implantación de un proceso de pruebas.
- Los actores del proceso productivo se deben acoplar al ciclo de vida unificado que se propone para llevar a cabo las pruebas de forma paralela al desarrollo de software.
- Es necesario que la organización cuente con un proceso claro y definido para el levantamiento de requisitos, dado que estos son un insumo indispensable para el diseño y ejecución de algunos de los tipos de pruebas propuestos en la técnica.
- La implantación del proceso de pruebas, requiere un proceso productivo maduro y refinado, cuyos cuellos de botella estén identificados y optimizados, con el fin de que se pueda obtener resultados positivos en cuanto a tiempos de prueba se refiere.
- Es necesario contar con un sistema de gestión de calidad para que las métricas generadas por la técnica sean administradas y se puedan utilizar para retroalimentar el proceso de pruebas.

3.3. Técnica propuesta

3.3.1. Definición de la Ingeniería de Requisitos

La ingeniería de requisitos es un proceso que comprende todas las actividades requeridas para crear y mantener un documento de requisitos del sistema. Existen cuatro actividades genéricas de alto nivel en el

proceso de ingeniería de requisitos. Estas son un estudio de factibilidad del sistema, la obtención y el análisis de requisitos, la especificación de éstos y su documentación y, finalmente, la validación. La Figura 2 ilustra la relación entre estas actividades. También muestra el documento que se produce en cada etapa del proceso de ingeniería de requisitos (IR).

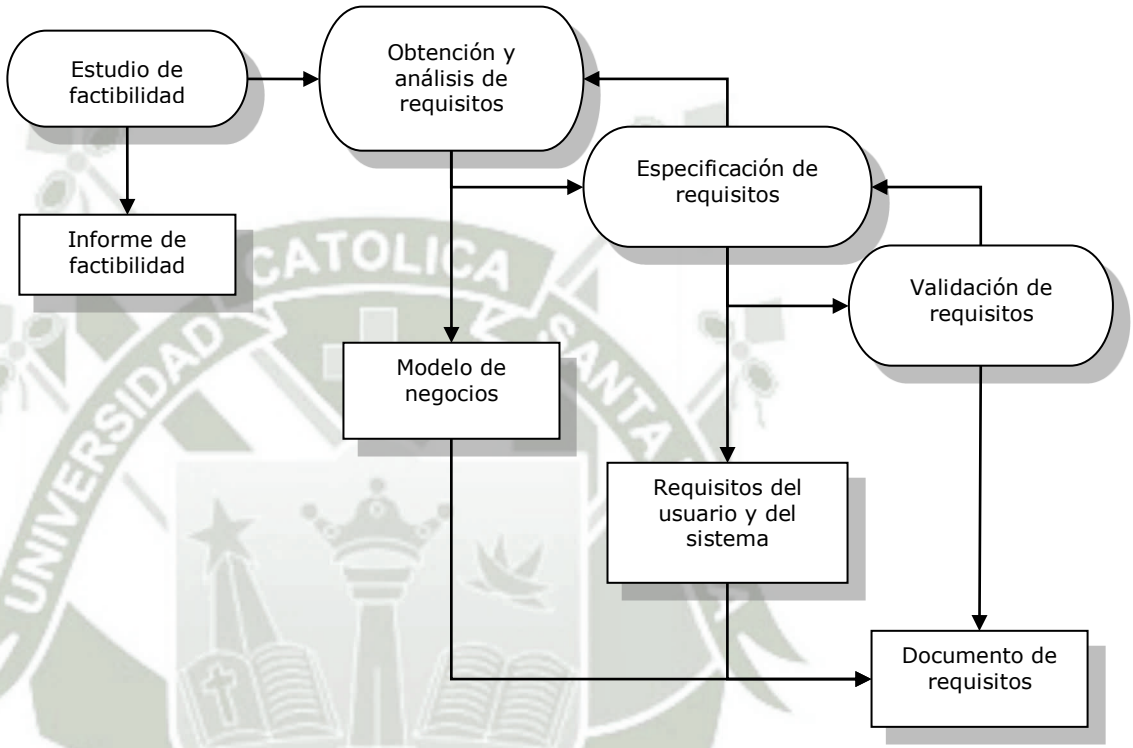


Figura Nº 2. El proceso de ingeniería de requisitos

A. Estudios de factibilidad

Para todos los sistemas nuevos, el proceso de ingeniería de requisitos empieza con un estudio de factibilidad. La entrada de éste es una descripción resumida del sistema y de cómo se utilizará dentro de una organización. El resultado del estudio es un informe que recomienda si es conveniente llevar a cabo la ingeniería de requisitos y el proceso de desarrollo del sistema.

Un estudio de factibilidad es un estudio corto y orientado a resolver varias preguntas:

1. ¿El sistema contribuye a los objetivos generales de la organización?
2. ¿El sistema se puede implementar utilizando la tecnología actual y con las restricciones de costo y tiempo?
3. ¿El sistema puede integrarse a otros que existen en la organización?

Aquí la cuestión crítica es si el sistema contribuye a los objetivos del negocio. Si no contribuye a estos, entonces no tiene valor real en el negocio.

B. Obtención y análisis de requisitos

En esta actividad, el personal del desarrollo técnico del software trabajara con los clientes y los usuarios finales del sistema para determinar el dominio de la aplicación, cuales servicios debe proveer el sistema, el desempeño requerido del sistema, las restricciones de hardware, y otros.

La obtención y análisis de requisitos incluyen diferentes tipos de personas de la organización. El termino *stakeholder* se utiliza para referirse a cualquier persona que tiene influencia directa o indirecta sobre los requisitos del sistema. Entre los *stakeholders* también son

los ingenieros que desarrollan o dan mantenimiento a otros sistemas relacionados, los administradores de negocio, los expertos en el dominio del sistema, los representantes de los trabajadores, y otros. En la Figura 3 se muestra un modelo genérico del proceso de obtención y análisis.

Cada organización tendrá su propia versión o instancia de este modelo general dependiendo de los factores locales, como la habilidad del personal, el tipo de sistema a desarrollar, los estándares utilizados, entre otros.

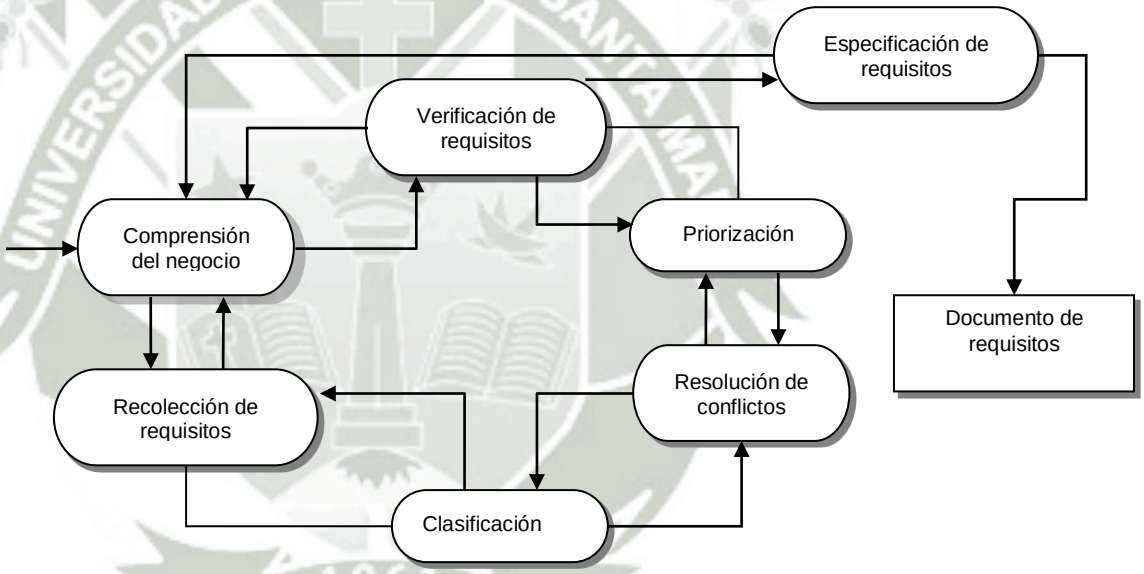


Figura 3. Proceso de obtención y análisis de requisitos

Las actividades del proceso son:

1. *Comprensión del dominio.* El analista debe desarrollar su propia comprensión del dominio de la aplicación.

2. *Recolección de requisitos.* Éste es el proceso de interactuar con los *stakeholders* del sistema para descubrir sus requisitos. Obviamente, la comprensión del dominio se desarrollará más durante esta actividad.

3. *Clasificación.* Esta actividad considera la recolección no estructurada de requisitos y los organiza en grupos coherentes.

4. *Resolución de conflictos.* De forma inevitable, cuando existen muchos *stakeholders* involucrados, los requisitos entrarán en conflicto. Esta actividad se refiere a encontrar y resolver estos conflictos.

5. *Priorización.* En cualquier conjunto de requisitos, alguno de ellos será más importante que los otros.

6. *Verificación de requisitos.* Los requisitos se verifican para descubrir si están completos, son consistentes y acordes con lo que realmente quieren los *stakeholders* del sistema.

La Figura 3 muestra que la obtención y análisis de requisitos es un proceso iterativo con retroalimentación continua de cada actividad a las otras actividades. El ciclo del proceso inicia con la comprensión del dominio y termina con la verificación de requisitos. La

comprensión de los requisitos por parte del analista mejora en cada vuelta de ciclo.

C. Técnicas para la obtención y análisis de requisitos

No existe un enfoque perfecto y universal aplicable a la obtención y el análisis de requisitos. Normalmente, es necesario utilizar varios de estos enfoques para desarrollar un entendimiento completo y un análisis de requisitos.

1. Obtención orientada a puntos de vista

   Para cualquier sistema grande o de mediano tamaño, existen diferentes tipos de usuarios finales. Muchos *stakeholders* tienen un cierto interés en los requisitos del sistema.

   Los diferentes puntos de vista de un problema consideran a este de diferentes formas. Sin embargo, sus perspectivas no son completamente independientes sino que se traslapan, por lo que tienen requisitos comunes.

   Un punto clave del análisis orientado a puntos de vista es que toma en cuenta la existencia de varias perspectivas y provee un marco de trabajo para descubrir conflictos en los requisitos propuestos por diferentes *stakeholders.*

Métodos diferentes tienen ideas diferentes de lo que significa un "punto de vista". Este se puede considerar como:

- *Una fuente o consumidor de datos.* En este caso, los puntos de vista son responsables de producir o consumir datos.

- *Un marco de trabajo de la representación*. En este caso, un punto de vista se considera un tipo particular del modelo del sistema.

- *Un receptor de servicios*. En este caso, los puntos de vista son externos al sistema y reciben servicios de él.

2. Escenarios

Normalmente, las personas encuentran más fácil dar ejemplos de la vida real que descripciones abstractas. Pueden comprender y criticar un escenario de cómo podría interactuar con un sistema de software. Los ingenieros de requisitos pueden utilizar la información obtenida de esta discusión para formular los requisitos reales del sistema.

Los escenarios pueden ser especialmente útiles para agregar detalle a un bosquejo de descripción de requisitos. Son descripciones de ejemplos de las sesiones de interacción. Cada escenario cubre uno o un número reducido de posibles interacciones. Diferentes formas

de escenarios se han desarrollado y proveen diferentes tipos de información en diferentes niveles de detalles del sistema.

El escenario inicia con un bosquejo de la interacción y, durante la obtención, se agregan detalles para crear una descripción completa de esa interacción. De forma general, un escenario incluye:

1. Una descripción del estado del sistema al inicio del escenario.
2. Una descripción del flujo normal de eventos en el escenario.
3. Una descripción de lo que puede ir mal y cómo manejarlo.
4. Información de otras actividades que se podrían llevar a cabo al mismo tiempo.
5. Una descripción del estado del sistema después de completar el escenario.

Es posible llevar a cabo de manera informal la obtención de requisitos basada en escenarios cuando los ingenieros de requisitos trabajan con los *stakeholders* en la identificación de escenarios y en la captura de detalles de dichos escenarios. De forma alternativa, se puede utilizar un enfoque más estructurado como los escenarios de eventos o los casos de uso.

3. Escenarios de eventos

Estos se utilizan para documentar el comportamiento del sistema cuando se le presenta eventos específicos. Cada evento de interacción distinto se documenta como un escenario de eventos

distinto. Los escenarios de eventos incluyen una descripción el flujo de datos y las acciones del sistema y documenta las excepciones que pueden surgir. Para ilustrar los escenarios de eventos, considere la Figura 4 que muestra el escenario del evento "iniciar transacción".

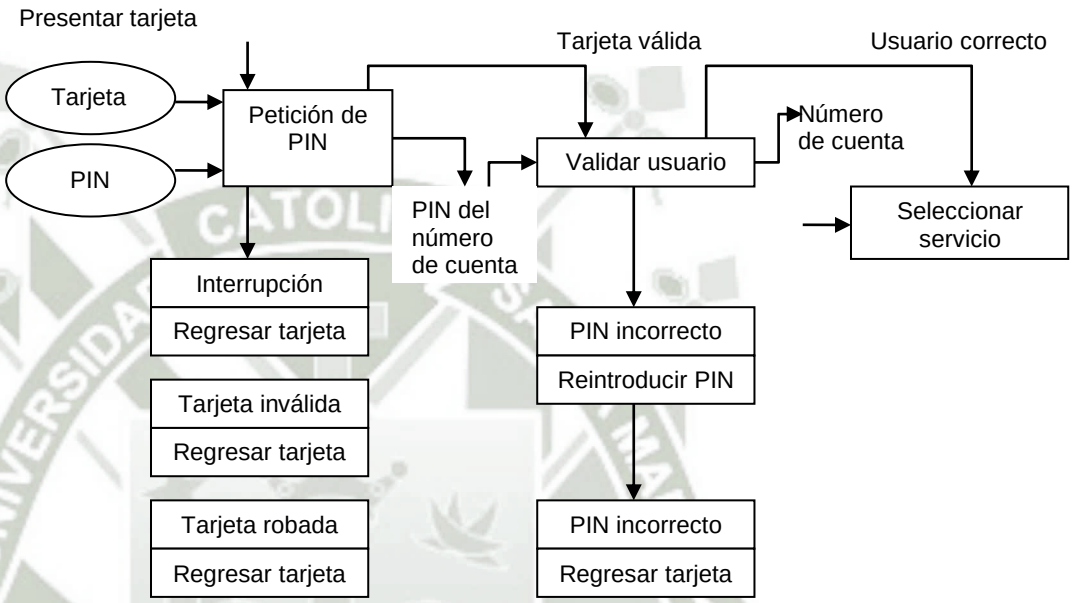


Figura 4. Escenario de eventos – Transacción de inicio

4. Casos de uso

Estos son una técnica que se basa en escenarios para la obtención de requisitos que se introdujeron por primera vez en el método *Objetory*.

Los actores en el proceso se representan como figuras delineadas y cada clase de interacción se representa como una elipse con su nombre. El conjunto de casos de uso representa todas las posibles interacciones que ocurrirán en los requisitos del sistema. Esto se ilustra, a medida de ejemplo, en la Figura 5. Algunas veces existe

confusión si un caso de uso es un escenario, un caso de uso encapsula un conjunto de escenarios en el que cada uno de éstos es un camino único a través de un caso de uso. En este caso habrá un escenario para la interacción normal y escenarios adicionales para las posibles excepciones.

Dentro de UML, los diagramas de secuencia se utilizan para agregar información a un caso de uso. Estos diagramas muestran a los actores involucrados en la interacción, los objetos del sistema como los que pueden interactuar y las operaciones asociadas con estos objetos como se muestra, a medida de ejemplo, en la Figura 5.

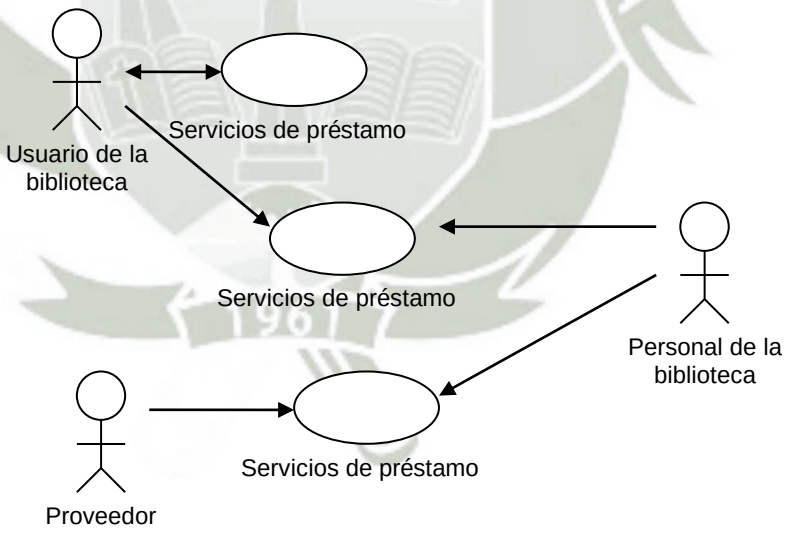


Figura 5. Casos de uso para una biblioteca

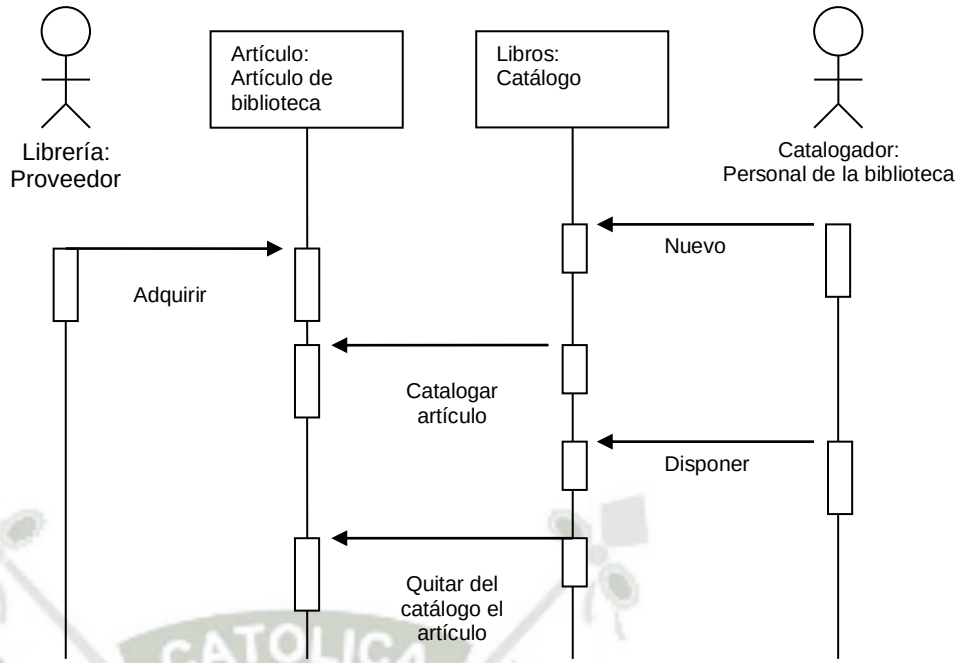


Figura 6. Diagrama de secuencia

5. Etnografía

La etnografía es una técnica de observación que se puede utilizar para entender los requisitos sociales y organizacionales. Un analista se sumerge por sí solo en el entorno laboral donde el sistema se utilizará. El trabajo diario se observa y se hacen notas de las tareas reales en las que los participantes están involucrados. El valor de la etnografía en que ayuda a descubrir los requisitos implícitos que reflejan los procesos reales más que los formales en los que la gente está involucrada.

A menudo, la gente encuentra muy difícil articular detalles de su propio trabajo debido a que están en segundo lugar en sus tareas diarias. Comprenden su propio trabajo pero no su relación con las demás tareas en la organización. Los factores sociales y organizacionales que afectan el trabajo, pero que no son obvios

para las personas, se clarifican cuando se presenta un observador imparcial.

La etnografía es especialmente efectiva para descubrir dos tipos de requisitos:

1. Los requisitos que se derivan de la forma en la que la gente trabaja realmente más que de la forma en la que las definiciones de los procesos establecen que debería trabajar.

2. Los requisitos que se derivan de la cooperación y conocimiento de las actividades de la gente.

La etnografía se puede combinar con la construcción de prototipo (ver Figura 7). La etnografía suministra información al desarrollo del prototipo de forma que se requieran menos ciclos de refinación.

Los estudios etnográficos pueden revelar los detalles de los procesos críticos que otras técnicas de obtención de requisitos a menudo olvidan. Sin embargo, puesto que se centran en el usuario final, este enfoque no es apropiado para descubrir los requisitos organizacionales o del dominio. No siempre puede identificar nuevas propiedades a agregar al sistema. Por lo tanto, la etnografía no es un enfoque completo para la obtención de requisitos y debe

utilizarse en conjunto con otros enfoques, como el análisis de casos de uso.

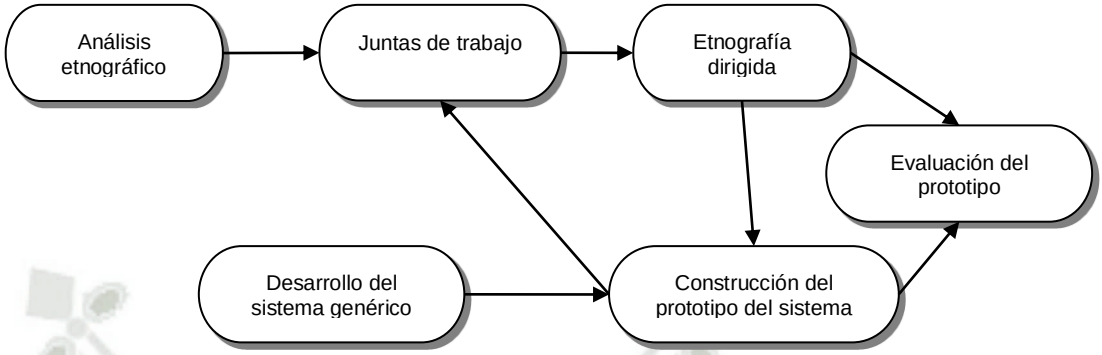


Figura Nº 7. Etnografía y construcción de prototipos para los requisitos

D. Validación de requisitos

Esta validación muestra que éstos son los que definen el sistema que el cliente desea. Tiene mucho en común con el análisis, ya que implica encontrar problemas con los requisitos. Sin embargo, son procesos distintos puesto que la validación comprende un bosquejo completo del documento de requisitos mientras que el análisis implica trabajar con requisitos incompletos.

Durante el proceso de validación de requisitos, se debe llevar a cabo diferentes tipos de verificación de requisitos en el documento de requisitos. Estas verificaciones incluyen:

1. *Verificación de validez*. Un usuario puede pensar que se necesita un sistema para llevar a cabo ciertas funciones. Sin embargo, el

razonamiento y el análisis identifican que se requieren funciones adicionales y diferentes.

2. *Verificaciones de consistencia*. Los requisitos en el documento no deben contradecirse.

3. *Verificaciones de integridad.* El documento de requisitos debe incluir requisitos que definan todas las funciones y restricciones propuestas por el usuario del sistema.

4. *Verificaciones de realismo*. Utilizando el conocimiento de la tecnología existente, los requisitos deben verificarse para asegurar que se pueden implementar.

5. *Verificabilidad*. Para reducir las discusiones entre el cliente y el contratista, los requisitos del sistema siempre deben redactarse de tal forma que sean verificables.

Existen varias técnicas de validación de requisitos que pueden utilizarse en conjunto o de forma individual:

1. *Revisiones de requisitos.* Los requisitos son analizados sistemáticamente por un equipo de revisores.

2. *Construcción de prototipos.* En este enfoque de validación, se muestra un modelo ejecutable del sistema a los usuarios finales y los clientes. Estos pueden hacer experimentos con este modelo para ver si cumple sus necesidades reales.

3. *Generación de caos de prueba.* De forma ideal, los requisitos deben poder probarse. Si las pruebas para éstos se consideran como parte del proceso de validación, esto a menudo revela los problemas en los requisitos. Si una prueba es difícil o imposible de diseñar, por lo regular esto significa que los requisitos serán difíciles de implementar y deberían ser considerados nuevamente.

4. *Análisis de consistencia automático*. Si los requisitos se expresan como un modelo del sistema en una notación estructurada o formal, entonces las herramientas CASE deben verificar la consistencia del modelo. Para verificar la consistencia, la herramienta CASE debe construir una base de datos de requisitos y después, utilizando las reglas del método o notación, verificar todos los requisitos en dicha base de datos. Un analizador de requisitos produce un informe de las inconsistencias recién descubiertas.

E. Administración de requisitos

La administración de requisitos es el proceso de comprender y controlar los cambios en los requisitos del sistema. El proceso de

administración de requisitos se lleva a cabo junto con los otros procesos de ingeniería de requisitos. La planeación comienza al mismo tiempo que la obtención de requisitos inicial y la administración activa de requisitos debe iniciar tan pronto como esté, lista la primera versión del documento de requisitos.

1. Requisitos duraderos y volátiles

   El desarrollo de requisitos de software centra su atención en las capacidades de éste, los objetivos del negocio y otros sistemas de la organización. Especificar el desarrollo de un sistema grande puede llevar varios años. Con el tiempo, el entorno del sistema y los objetivos del negocio seguramente cambiará. Por lo tanto, los requisitos deben evolucionar para reflejar esto. Desde una perspectiva evolutiva, los requisitos son de dos clases:

   a) *Requisitos duraderos.* Éstos son relativamente estables que se derivan de la actividad principal de la organización y que están relacionados directamente con el dominio del sistema.

   b) *Requisitos volátiles.* Éstos cambiarán probablemente durante el desarrollo del sistema o después de que éste se haya puesto en operación.

   Los requisitos volátiles pueden ser de las siguientes clases, como se muestra en la Tabla 1.

Tabla 1

Clasificación de requisitos volátiles

Tipos de requisitos	Descripción
Requisitos mutantes	Requisitos que cambian debido a los cambios en el ambiente en el que opera la organización.
Requisitos emergentes	Requisitos que emergen al incrementarse la comprensión del cliente en el desarrollo del sistema. El proceso de diseño puede revelar requisitos emergentes nuevos.
Requisitos consecutivos	Requisitos que son resultado de la introducción del sistema de cómputo
Requisitos de compatibilidad	Requisitos que dependen de sistemas particulares o procesos de negocios dentro de la organización. Cuando estos últimos cambian, los requisitos de compatibilidad del sistema contratado o a entregar también pueden cambiar.

Fuente: (Pressmann, 2005)

2. Planeación de la administración de requisitos

Ésta es una primera etapa esencial del proceso de administración de requisitos. La administración de requisitos es muy cara y, para cada proyecto, la etapa de planeación establece el nivel de detalle necesario en la administración de requisitos. Durante la etapa de administración de requisitos se tiene que decidir sobre:

a) *La identificación de requisitos.* Cada requerimiento se debe identificar de forma única de tal forma que puedan entrar en referencia cruzada con otros requisitos de manera que pueda utilizarse en las evaluaciones de rastreo.

b) *Un proceso de administración del cambio.* Éste es el conjunto de actividades que evalúa el impacto y costo de los cambios.

c) *Políticas de rastreo*. Éstas definen la relación entre requisitos y la de éstos en el diseño del sistema que se debe registrar y la manera en que estos registros se deben mantener.

d) *Ayuda de herramientas CASE*. La administración de requisitos comprende el procesamiento de grandes cantidades de información de los requisitos.

3. Administración del cambio de los requisitos

Esta administración se aplica a todos los cambios propuestos en los requisitos. La ventaja de utilizar un proceso formal para administrar el cambio es que todos los cambios propuestos son tratados de forma consistente y que los cambios en el documento de requisitos se hacen de forma controlada. Existen tres etapas principales en un proceso de administración de cambio:

1. *Análisis de problema y especificación del cambio.*
2. *Análisis del cambio y costeo.*
3. *Implementación del cambio.*

3.3.2. Definición de roles

Los actores involucrados directamente en el área de pruebas deben ser claramente definidos mediante una determinación de roles con el fin maximizar las habilidades del factor humano y permitir la fácil asignación de tareas y responsabilidades al interior. Así clasificación de roles y actores está dado según el conocimiento, la responsabilidad y las tareas que se deben desempeñar en dicho cargo, así:

- Coordinador de pruebas: encargado de liderar el grupo de pruebas, constituye el puente principal de comunicaciones entre el personal de aseguramiento de la calidad del producto y los líderes de otras áreas de procesos y proyectos en general.
- Diseñador de pruebas: Dado que el diseño de las pruebas se constituye como la actividad más compleja y que lleva más tiempo a través de todo los procesos de aseguramiento de la calidad del producto, el desempeño de éste rol constituye uno de los más importantes, además que se desagrega con el fin de evitar la subestimación de habilidades y conocimientos del personal humano. Así pues existen tres tipos de diseñadores de pruebas que se determinan según el tipo y la complejidad de las pruebas que diseñan; éstos son: diseñador de alto, medio y bajo nivel.
- Probador: Ejecuta casos de prueba ya aprobados y reporta los errores al área de producción.

3.3.3. Definición de las pruebas de software

Las pruebas de software pueden ser de muchos tipos, además de pruebas estándares en el mercado del software existen pruebas más especializas que agregan confiabilidad y estabilidad a largo plazo al producto y en algunos casos mejor rendimiento y eficiencia. Las pruebas que son obligatorias para un aplicativo son las funcionales que se apoyan directamente sobre los requisitos y todo aquel artefacto que los exprese, defina o clarifique.

Generalmente las prueba funcionales son basadas en casos de uso y se orientan a detectar posibles incoherencias con lo que el cliente necesita y errores de programación y diseño en general. Existen otras pruebas que buscan determinar los límites del aplicativo y sus debilidades, éstas son llamadas pruebas de tensión (stress) y pruebas de rendimiento (performance). Finalmente existen prueba que se especializan en descubrir que tan bueno es el producto de software en tareas específicas como por ejemplo: pruebas de instalación, seguridad, documentación, mantenimiento, recuperación, entre otras.

La elección de qué pruebas realizar y cuál es el criterio de aceptación del producto constituye uno de los factores más importante a la hora de realizar pruebas, y para esto se deben evaluar tanto los riesgos como las necesidades del aplicativo mismo, según sus características propias y las del usuario final.

3.3.4. Proceso de la técnica

Las metodologías adoptadas para crear filtros en diferentes puntos del proceso de producción son: verificación y validación, soportado a través del mecanismo de pruebas por pares, en las cuales existen tres actores: dos probadores y un árbitro. Las etapas en las cuales se propone introducir dichos filtros son:

- Etapa de Análisis y diseño: Esta fase es crucial para obtener un producto conforme con los requisitos y expectativas del interesado, es por ello que es necesario asegurar que el software es una representación correcta de estos artefactos. El artefacto más importante y que apunta a las pruebas de funcionalidad son los casos de uso, otros no menos importantes, pero si menos utilizados son los diagramas de transición de estados, de secuencias y de actividades; estos además de ayudar al modelamiento de la solución permiten utilizar pruebas basadas en novedosas técnicas para determinar errores de graves consecuencias en el sistema. En ésta etapa se debe definir un plan de pruebas que contemple: análisis del dominio, análisis del aplicativo desde el punto de vista de las pruebas de software, delimitación de alcances de las pruebas, definición de los tipos de pruebas a diseñar y ejecutar, detección y manejo de riesgos, creación de cronograma con tareas específicas e hitos, determinación de los recursos mediante la estimación entre otros.
- Etapa de Codificación: En este punto se deben capturar los errores superficiales que se comenten generalmente por descuidos y que por lo general no alteran la funcionalidad del producto, estos errores se

refieren a interfaces a nivel de comportamiento, y ayudan a obtener un producto de excelente calidad. También se deben realizar comprobaciones del uso de estándares de programación ya definidos por la organización. Las pruebas unitarias mediante técnicas de caja blanca son seguidas de pruebas por pares mediante técnicas de caja negra que ayudan a obtener un grado de madurez mayor al área de pruebas. El diseño de casos de prueba se debe iniciar en ésta etapa antecedido de capacitaciones en el dominio a los diseñadores y probadores encargados del proyecto.

- Etapa de pruebas: Los subprocesos internos que se ejecutan en ésta etapa son:
  - Diseño de casos de prueba: por módulos y por tipos de pruebas siguiendo lo establecido en el plan de pruebas. La reutilización hace parte fundamental en éste subproceso ya que permite no solo madurar pruebas específicas a través del tiempo, sino también ahorro de tiempo y recursos. Se debe registrar cada caso de prueba en una herramienta de administración de casos de prueba, con el fin de automatizar ciertas tareas, reutilizar fácilmente y llevar seguimiento. Al final de éste subproceso se debe verificar mediante listas de chequeo que los casos de prueba diseñados sean conformes y cumplan con los objetivos y alcances de las pruebas.
  - Ejecución de casos de prueba y reporte de errores: la ejecución de casos de pruebas se realiza siguiendo los lineamientos de los casos de prueba diseñados y concluye con el reporte de errores

encontrados en el aplicativo; para tal fin se pueden utilizar formatos o herramientas administradoras de errores o bugtracker con el fin de automatizar y facilitar ciertas tareas, además de permitir el seguimiento de todo lo concerniente con el reporte y corrección de los mismos. Al final de éste subproceso se debe verificar mediante listas de chequeo que los errores reportados si sean en realidad errores y que su reporte sea suficientemente claro y bien documentado para su entendimiento y posterior solución.

- Regresiones y aprobación: después del reporte de los errores, la corrección de los mismos implica volver a ejecutar los casos de prueba fallidos con el fin de verificar que dichos errores en efecto se solucionaron y no afectaron otros aspectos locales del software. Antes de realizar una aprobación del producto de software, se debe realizar un análisis en el cual se tengan en cuenta las regresiones realizadas al software, los errores y las correcciones hechas, debido a que su impacto puede ser alto en otros puntos de aplicativo globalmente, lo cual podría suponer volver a ejecutar las pruebas a otros módulos o el diseño de pruebas adicionales en partes específicas del aplicativo.

- Etapas de implantación, pruebas del cliente, soporte y mantenimiento: cuando se realiza la instalación en el ambiente de preproducción y posteriormente en el ambiente de producción del cliente, se deben utilizar listas de chequeos que permitan verificar

que la instalación es conforme y se debe documentar el estado final de la instalación con el fin de tener un apoyo para la posible determinación de la fuente de un error en la etapa de soporte y mantenimiento. En ambientes de preproducción generalmente se realizan algunos reportes de cambios y errores los cuales deben ser manejados según su criticidad y complejidad, lo cual puede llevar a realizar nuevas pruebas al software o volver a ejecutar muchas de las pruebas realizadas en etapas anteriores.

3.3.5. Elección de la técnica para determinar el conjunto de datos

- Aleatoria
- División de clases equivalentes
- Valores límite
- Transición de estado
- Suposición de error

3.3.6. Optimización de los casos de prueba

- El diseño de los casos de prueba, que es una de las actividades que más tiempo consumen, comienza tan pronto los artefactos de análisis y diseño son construidos.
- La obtención del conocimiento del dominio necesario para diseñar y ejecutar algunas pruebas debe ser obtenido antes de empezar la etapa de codificación, teniendo presente que en el trascurso del desarrollo es muy probable que se debe obtener información adicional del dominio.

- Las listas de chequeo de verificación y validación deben entrar como insumo al proceso de pruebas, para evitar redundancias en las pruebas y permitir la realimentación del área de pruebas.
- Las solicitudes de prueba deben generarse por eventos, estos eventos están asociados con la terminación de la codificación de una nueva funcionalidad o modulo, el reporte de un cambio, un error o un requisito nuevo en etapas de mantenimiento del producto.
- Cuando se reporta un error este debe ser corregido por el área de producción y generar una prueba llamada regresión para verificar que la corrección elimino el error y no introdujo más. El determinar si ocurrieron más errores es bastante difícil, puesto que la única solución sería probar de nuevo todo el aplicativo, esto último es impráctico y no trae buenos resultados, por lo tanto se debe considerar el juicio experto apoyado por matrices de trazabilidad para determinar que otros módulos o funcionalidades fueron afectados por la solución y así realizar las pruebas necesarias a estos.
- La cobertura para pruebas de caja blanca esta medida en líneas de código probadas, al ser esta una metodología basada en pruebas de caja negra es necesario medir la cobertura en otros términos. Un análisis en términos de pruebas básicas que cualquier aplicativo debe tener, número de casos de prueba diseñados y ejecutados, y pesos relativos a los diferentes tipos de pruebas permiten obtener a

través de relaciones lineales el porcentaje de cobertura de las pruebas.

3.3.7. Medidas asociadas

El área de pruebas acoplada al plan de estimación de la compañía puede beneficiarse, debido a que un estudio de la cantidad de errores inyectados por aplicativo asociados a su nivel de complejidad y el de sus errores puede llevar a generar una estadística de los errores esperados según el tamaño del software.

Existen técnicas asociadas a modelos de estimación que permiten calcular el número de defectos esperados en una pieza de software medida, uno de los más utilizados es COQUALMO que hace parte de COCOMO II. Esto último suele ser útil a la hora de decidir cuánto se debe probar y a qué nivel. Las medidas asociadas se encuentran enmarcadas dentro de los siguientes factores:

Tabla 2

Medidas asociadas a las pruebas de software

Factor	Descripción
Tipo de licencia y precio	El tipo de licencia delimita la forma de uso y está directamente relacionado con el precio, ya que dependiendo de las capacidades o límites del aplicativo los precios suben o bajan según sea el caso. Generalmente en éste tipo de aplicaciones el precio lo delimita el número máximo de usuarios concurrentes que es posible emular.
Acceso al código fuente	Posibilidad de acceder al código fuente de la herramienta y realizar modificaciones en ella. Generalmente esto depende del tipo de licencia que cada una de ellas tenga.
Plataforma	Sistemas soportados para la instalación de la herramienta que en términos precisos se traduce en los sistemas operativos soportados o el

	lenguaje de programación usado para la construcción de la herramienta que en algunos casos se traduce en portabilidad entre sistemas, tal caso son las herramientas programadas en java, las cuales generalmente son portables entre sistemas Windows, Linux, UNIX, Solaris, entre otros.
Requisitos de hardware	Requisitos del sistema a nivel de hardware que se requieren para que la herramienta se ejecute correctamente. Se refiere generalmente al tipo de procesador, memoria RAM y disco duro.
Pruebas programadas	Permite programar el inicio, las paradas y el fin de un plan de pruebas específica.
Número de informes nativos	Mientras más informes predeterminados tengan, la facilidad de interpretación de resultados será más alta y por ende el tiempo de la prueba se reducirá.
Diseño de informes personalizados	Las técnicas generalmente traen consigo ya por defecto informes específicos para el análisis de los resultados de las pruebas, sin embargo, algunas más poderosas permiten la manipulación de variables propias en las pruebas y su uso en la construcción de gráficas personalizadas que permiten un análisis específico y acoplado al ambiente determinado de las pruebas.
Monitoreo de Bases de Datos	Propias de la aplicación que se conectan directamente al motor de base de datos que utiliza la aplicación probada y permite visualizar su comportamiento en términos de rendimiento y uso de recursos a medida que la prueba es ejecutada.
Pruebas de segmentos múltiples	Se refiere a la posibilidad de realizar una prueba desde distintos puntos de una arquitectura de red, ya sea aplicando el mismo plan de pruebas o uno distinto. Se usa tanto para evaluar el comportamiento de la aplicación web probada bajo la carga desde distintos nodos de una red, como para potenciar más la carga de trabajo, ya que en algunos casos el ambiente en el cual se ejecuta la herramienta para realizar la prueba se puede sobrecargar y arrojar resultados incoherentes.
Pruebas funcionales	En herramientas muy especializadas, permiten inicialmente realizar pruebas funcionales basadas en requisitos, generalmente programables por medio de script. En algunas de ellas se pueden realizar pruebas de carga basadas en las pruebas funcionales programadas, permitiendo un ahorro de tiempo y un desempeño más realista durante las pruebas.
Posibilidad de extensiones	Funcionalidad que permite incrementar las funcionalidades de la aplicación mediante

	adiciones, script o subprogramas.
Escalabilidad de usuarios	Funcionalidad de la herramienta de generar un número de usuarios virtuales de acuerdo a los recursos específicos de la máquina en la cual se ejecuta la prueba. Generalmente éste factor depende del número, tamaño y complejidad del script de prueba.

Fuente: Elaboración propia

4.1 Introducción

La Cooperativa de Ahorro y Créditos Alto Selva Alegre actualmente cuenta con un sistema en el cual se administran los socios y cuáles de estos tienen accesos a créditos; de forma separada y en otro programa se lleva la evaluación para poder determinar si un socio es aceptado para obtener un crédito.

Este método de trabajo hace que los gestores de créditos tengan que realizar doble trabajo, lo cual implica una demora en su atención al cliente cuando este solicita un crédito. El Modulo de Evaluación de Créditos agilizará la evaluación de dichas solicitudes teniendo como base el historial crediticio del socio que se encuentra almacenado en la base de datos de la cooperativa. De esta manera se agilizaría la obtención de los resultados de las solicitudes de créditos. Este sistema servirá de base para llevar a cabo la comprobación de la técnica propuesta.

4.2. Aplicación de la técnica

4.2.1 Educción de requisitos

ED-0001	**Ingreso al Sistema**
Versión	0.4
Autores	Gómez Herrera, César Eduardo Phoco López, Marcos
Fuentes	Llerena Puma, Danilo Urquizo Concha, Fernando
Descripción	La persona que hará uso del sistema lo hará después de ingresar su usuario y contraseña.
Importancia	Vital
Estado	
Comentarios	El acceso es por cuentas de usuario.

ED-0002	Búsqueda de socios
Versión	0.4
Autores	Gómez Herrera, César Eduardo Phoco López, Marcos
Fuentes	Llerena Puma, Danilo Urquizo Concha, Fernando
Descripción	El Software de la Cooperativa, debe de verificar la existencia del Socio, realizando una búsqueda por nombre o código del socio.
Importancia	Vital
Estado	
Comentarios	Datos que se toman : Nombres y Apellidos / código de socio

ED-0003	Análisis de historial crediticio
Versión	0.4
Autores	Gómez Herrera, César Eduardo Phoco López, Marcos
Fuentes	Llerena Puma, Danilo Urquizo Concha, Fernando
Descripción	La aplicación deberá poder realizar un análisis del historial crediticio y así poder mostrar si es viable o no realizar un crédito a dicho socio.
Importancia	Vital
Estado	
Comentarios	El análisis en función a los 3 últimos créditos y a los días de mora

ED-0004	Impresión de evaluación crediticia
Versión	0.4
Autores	Gómez Herrera, César Eduardo Phoco López, Marcos
Fuentes	Llerena Puma, Danilo Urquizo Concha, Fernando
Descripción	La aplicación deberá poder imprimir el reporte generado de la evaluación del historial crediticio de socio.
Importancia	Vital
Estado	
Comentarios	Ninguno

ED-0005	Manual de usuario
Versión	0.4
Autores	Gómez Herrera, César Eduardo Phoco López, Marcos
Fuentes	Llerena Puma, Danilo Urquizo Concha, Fernando
Descripción	La aplicación deberá contar con un manual de usuario.
Importancia	Vital
Estado	
Comentarios	Ninguno

ED-0006	Manual técnico
Versión	0.4
Autores	Gómez Herrera, César Eduardo Phoco López, Marcos
Fuentes	Llerena Puma, Danilo Urquizo Concha, Fernando
Descripción	La aplicación deberá contar con un manual técnico.
Importancia	Vital
Estado	

Comentarios	Ninguno

ED-0007	**Registrar usuarios**
Versión	0.4
Autores	Gómez Herrera, César Eduardo Phoco López, Marcos
Fuentes	Llerena Puma, Danilo Urquizo Concha, Fernando
Descripción	La aplicación deberá crear usuarios (socios) teniendo en cuenta los parámetros: identificador, nombre y apellidos.
Importancia	Vital
Estado	
Comentarios	Ninguno

ED-0008	**Modificar usuarios**
Versión	0.4
Autores	Gómez Herrera, César Eduardo Phoco López, Marcos
Fuentes	Llerena Puma, Danilo Urquizo Concha, Fernando
Descripción	La aplicación deberá permitir la modificación de los atributos del socio.
Importancia	Vital
Estado	
Comentarios	Ninguno

ED-0009	**Eliminar usuarios**
Versión	0.4
Autores	Gómez Herrera, César Eduardo Phoco López, Marcos
Fuentes	Llerena Puma, Danilo Urquizo Concha, Fernando
Descripción	En la aplicación permite eliminar cualquier socio.
Importancia	Vital
Estado	
Comentarios	Ninguno

4.2.2 Elicitación de requisitos

EL – 0001	**Login**
Versión	0.2
Descripción	La aplicación permitirá el uso de la misma, si el usuario previamente registrado ingresa nombre de usuario y contraseña válidos.
Prioridad del requisito	Alta
Dependencia	**ED-0001**
Datos Específicos	• Identificador de usuario • Password
Comentarios	Ninguno

EL – 0002	Búsqueda de Socio por Apellido
Versión	0.2
Descripción	La aplicación permitirá realizar la búsqueda del socio por medio de su nombre y apellidos.
Prioridad del requisito	Alta
Dependencia	**ED-0002**
Datos Específicos	• Nombre y apellido del socio
Comentarios	Como resultado de la búsqueda se mostrara una lita de socios cuyos datos correspondan a la búsqueda ingresada

EL – 0003	Búsqueda de Socio por Código
Versión	0.2
Descripción	La aplicación permitirá realizar la búsqueda del socio por medio de un código único.
Prioridad del requisito	Alta
Dependencia	**ED-0002**
Datos Específicos	• Código del socio
Comentarios	Ninguno

EL-0004	Consulta de historial crediticio
Versión	0.2
Descripción	La aplicación deberá poder consultar el historial crediticio de los socios para poder asignarles un nivel y mostrar si es viable o no un crédito.
Dependencia	Alta
Requisitos Asociados	**ED-0003**
Datos Específicos	Ninguno
Comentarios	El nivel será dado en referencia a los siguientes datos Excelente: cuando no tiene ninguna mora Bueno: cuando tiene moras de 1 a 10 días Regular: cuando tienen moras de 11 a 30 días Dudoso: cuando tienen moras de 31 a más 59 días Riesgoso: cuando tienen moras de 60 a más días

EL – 0005	Impresión de historial crediticio
Versión	0.2
Descripción	La aplicación deberá poder imprimir los últimos 3 créditos del socio, así como también el nivel asociado a este.
Prioridad del requisito	Alta
Requisitos Asociados	**ED-0004**
Datos Específicos	Ninguno
Comentarios	Ninguno

EL – 0006	Manual de Usuario
Versión	0.2
Descripción	La aplicación contará con un manual de usuario el cual facilitará su uso. Todos los usuarios tienen acceso al Manual de Usuario
Prioridad del requisito	Alto
Requisitos Asociados	**ED-0005**
Datos Específicos	Ninguno
Comentarios	Ninguno

EL – 0007	Manual técnico
Versión	0.1
Descripción	En este manual debe verificarse las características técnicas propias para el mantenimiento de la aplicación
Prioridad del requisito	Alto
Requisitos Asociados	**ED-0006**
Datos Específicos	Ninguno
Comentarios	Ninguno

EL – 0008	**Registrar Usuario**
Versión	0.2
Descripción	La aplicación permitirá la creación de nuevo usuarios (socio).
Prioridad del requisito	Alta
Dependencia	**ED-0001**
Datos Específicos	• Identificador de usuario • Password • Nombre y apellidos de usuario
Comentarios	Esta tarea será realizada por el administrador del sistema

EL – 0009	**Modificar Usuario**
Versión	0.2
Descripción	La aplicación permitirá la modificación de usuarios
Prioridad del requisito	Alta
Dependencia	**ED-0001**
Datos Específicos	• Código del usuario • Identificador de usuario • Password • Nombre y apellidos del usuario
Comentarios	Esta tarea será realizada por el administrador del sistema

EL – 0010	**Eliminar Usuario**
Versión	0.2
Descripción	La aplicación permitirá la eliminación usuarios
Prioridad del requisito	Alta
Dependencia	**ED-0001**
Datos Específicos	• Código de usuario
Comentarios	Esta tarea será realizada por el administrador del sistema

4.2.3 Especificación de requisitos de software

4.2.3.1 Definición de Stakeholders

Participante [STK-0001]	**Llerena Puma, Danilo (Jefe de Gestor)**
Organización	Cooperativa de Ahorro y Créditos Alto Selva Alegre
Rol	Cliente
Es desarrollador	No
Es cliente	Si
Es usuario	Si
Comentarios	Persona encargada de evaluar las solicitudes de crédito, dar informes a los clientes, afiliar nuevos socios, y controlar a los gestores

Participante [STK-0002]	**Urquizo Concha, Fernando (Gestor)**
Organización	Cooperativa de Ahorro y Créditos Alto Selva Alegre
Rol	Cliente
Es desarrollador	No
Es cliente	Si
Es usuario	Si
Comentarios	Persona encargada de evaluar las solicitudes de crédito, dar informes a los clientes, afiliar nuevos socios.

Participante [STK-0003]	**Jhonatan Zegarra (Administrador de sistemas)**
Organización	Cooperativa de Ahorro y Créditos Alto Selva Alegre (ASACOOP)
Rol	Cliente
Es desarrollador	No
Es cliente	Si
Es usuario	Si
Comentarios	Persona encargada administrar los programas usados por la ASACOOP

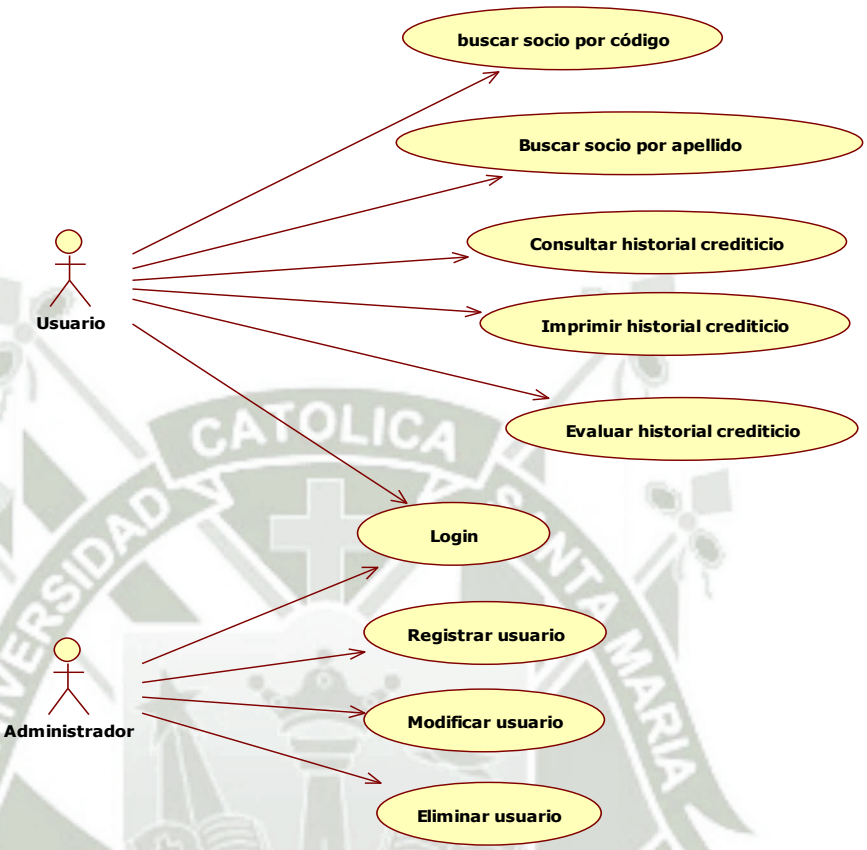
4.2.3.2 Definición de los Actores

ACT-0001	**Usuario**
Versión	0.1
Autores	Grupo de Analistas
Fuente	[STK-0001] [STK-0002] [STK-0003]
Descripción	Este actor hará uso del Sistema de Evaluación de Créditos.
Comentarios	Persona que hará uso de la aplicación [STK-0001] y [STK-0002]

ACT-0002	**Administrador**
Versión	0.1
Autores	Grupo de Analistas
Fuente	[STK-0001] [STK-0002]
Descripción	Este actor hará uso del Sistema de Evaluación de Créditos
Comentarios	Persona encargada [STK-0003]

4.2.3.3 Reportes de casos de uso esenciales

Evaluación de crédito



CU – 0001	Login
Descripción	Para el ingreso al sistema es obligatorio que cada usuario brinde su nombre de usuario y password.
Dependencias	**EL-0001**
Actores	**ACT-0001 y ACT-0002**
Comentarios	Ninguno

CU – 0002	Buscar socio por apellido
Descripción	El gestor podrá buscar a un socio o cliente por el Apellido del socio.
Dependencias	**EL-0002**
Actores	**ACT-0001**
Comentarios	Ninguno

CU – 0003	Buscar socio por Código
Descripción	El gestor podrá buscar a un socio o cliente por el código de socio
Dependencias	**EL-0003**
Actores	**ACT-0001**
Comentarios	Ninguno

CU – 0004	Consultar historial Crediticio
Descripción	El gestor tendrá la opción de consultar el historial crediticio dado que está registrado en el sistema.
Dependencias	**EL-0002,EL-0003**
Actores	**ACT-0001**
Comentarios	Ninguno

CU – 0005	Evaluar historial Crediticio
Descripción	El gestor podrá evaluar el historial crediticio de los socios para poder asignarles un nivel y mostrar si es viable o no un crédito.
Dependencias	**EL-0004**
Actores	**ACT-0001**
Comentarios	Ninguno

CU - 0006	Imprimir Historial Crediticio
Descripción	El gestor tendrá la opción de imprimir el historial que se le mostrara en la ventana de consulta de historial crediticio.
Dependencias	**EL-0005**
Actores	**ACT-0001**
Comentarios	Ninguno

CU – 0007	Registrar usuario
Descripción	El administrador podrá crear nuevos usuarios del sistema.
Dependencias	**EL-0008**
Actores	**ACT-0002**
Comentarios	Ninguno

CU – 0008	Modificar usuario
Descripción	El administrador podrá realizar modificaciones a los usuarios registrados en el sistema.
Dependencias	**EL-0009**
Actores	**ACT-0002**
Comentarios	Ninguno

CU – 0009	Eliminar usuario
Descripción	El administrador podrá eliminar usuarios registrados en el sistema.
Dependencias	**EL-0010**
Actores	**ACT-0002**
Comentarios	Ninguno

4.2.3.4 Requerimientos funcionales:

FRQ– 0001	Login
Versión	0.2
Autores	Gómez Herrera, César Eduardo Yauri Castillo, Jorge Iván
Fuentes	**[STK-0001], [STK-0002] Y [STK-0003]**
Dependencias	Ninguna
Descripción	La aplicación deberá permitir el acceso restringido, se permite el acceso únicamente si el usuario se identifica utilizando un nombre de usuario y password. El usuario ha debido ser previamente registrado.
Importancia	Vital
Urgencia	Inmediata
Estado	Pendiente de verificación

FRQ– 0002	Buscar Socio
Versión	0.2
Autores	Yauri Castillo, Jorge Iván
Fuentes	**[STK-0001], [STK-0002] Y [STK-0003]**
Dependencias	**FRQ-0001**
Descripción	La aplicación deberá permitir la búsqueda de un socio por apellido o

	código.
Importancia	Vital
Urgencia	Inmediata
Estado	Pendiente de verificación

FRQ– 0003	**Consultar el historial de Crédito**
Versión	0.2
Autores	Yauri Castillo, Jorge Iván
Fuentes	**[STK-0001], [STK-0002] Y [STK-0003]**
Dependencias	**FRQ-0001**
Descripción	La aplicación deberá permitir la consulta del historial crediticio de un socio.
Importancia	Vital
Urgencia	Inmediata
Estado	Pendiente de verificación

FRQ– 0004	**Evaluar Historial Crediticio**
Versión	0.2
Autores	Yauri Castillo, Jorge Iván
Fuentes	**[STK-0001], [STK-0002] Y [STK-0003]**
Dependencias	**FRQ-0003**
Descripción	La aplicación evaluara el historial crediticio de socio calificándolo entre 5 categorías que son: Excelente: cuando no tiene ninguna mora Bueno: cuando tiene moras de 1 a 10 días Regular: cuando tienen moras de 11 a 30 días Dudoso: cuando tienen moras de 31 a más 59 días Riesgoso: cuando tienen moras de 60 a más días
Importancia	Vital
Urgencia	Inmediata
Estado	Pendiente de verificación

FRQ– 0005	**Imprimir historial crediticio**
Versión	0.2
Autores	Yauri Castillo, Jorge Iván
Fuentes	**[STK-0001], [STK-0002] Y [STK-0003]**

Dependencias	**FRQ-0003**
Descripción	La aplicación deberá permitir la impresión del historial crediticio y adjuntarlo como documento en caso procesa la solicitud de crédito del socio.
Importancia	Vital
Urgencia	Inmediata
Estado	Pendiente de verificación

FRQ– 0007	**Registrar usuario**
Versión	0.1
Autores	Yauri Castillo, Jorge Iván
Fuentes	**[STK-0001], [STK-0002] Y [STK-0003]**
Dependencias	**FRQ-0001**
Descripción	La aplicación deberá permitir el registro de un nuevo usuario del sistema.
Importancia	Vital
Urgencia	Inmediata
Estado	Pendiente de verificación

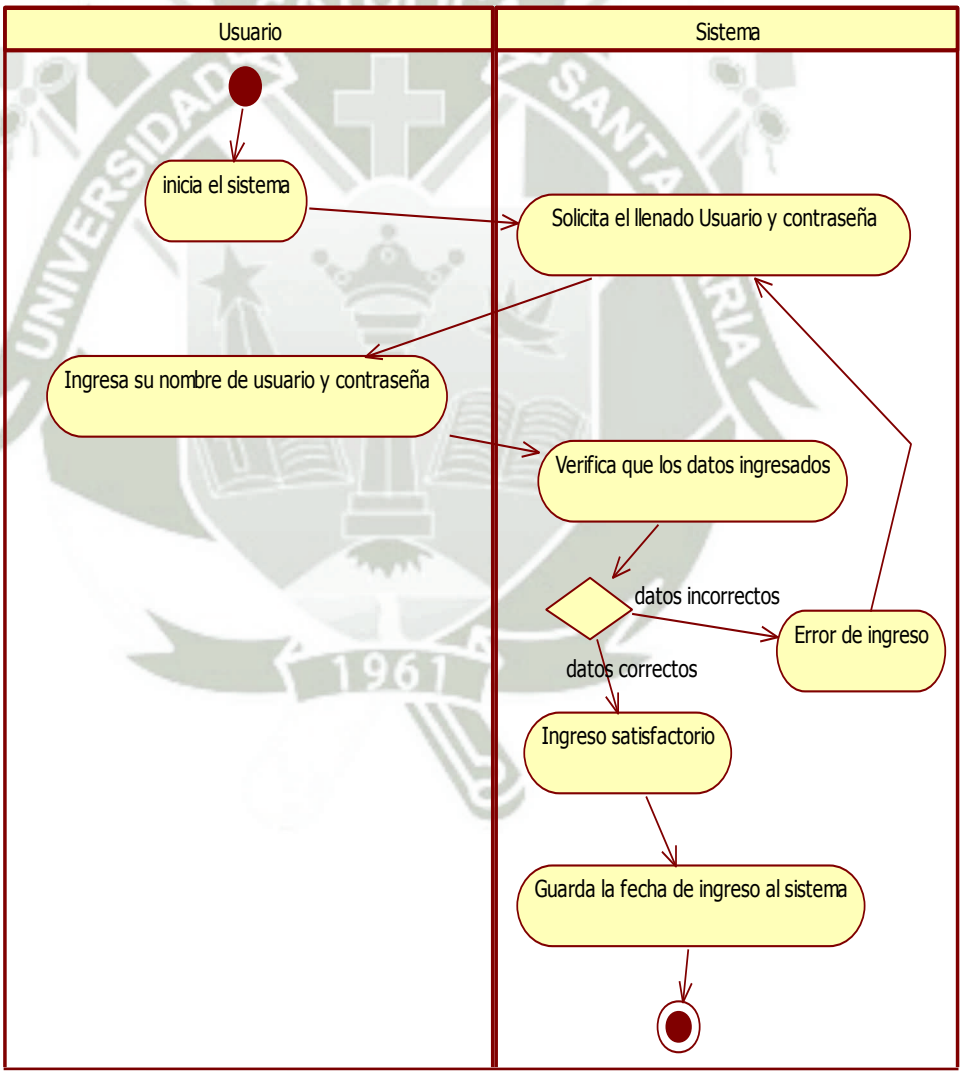
FRQ– 0008	**Modificar usuario**
Versión	0.1
Autores	Yauri Castillo, Jorge Iván
Fuentes	**[STK-0001], [STK-0002] Y [STK-0003]**
Dependencias	**FRQ-0001**
Descripción	La aplicación deberá permitir la modificación de un usuario (actualización de datos pertenecientes a un usuario registrado en el sistema)
Importancia	Vital
Urgencia	Inmediata
Estado	Pendiente de verificación

FRQ– 0009	**Eliminar usuario**
Versión	0.1
Autores	Yauri Castillo, Jorge Iván
Fuentes	**[STK-0001], [STK-0002] Y [STK-0003]**

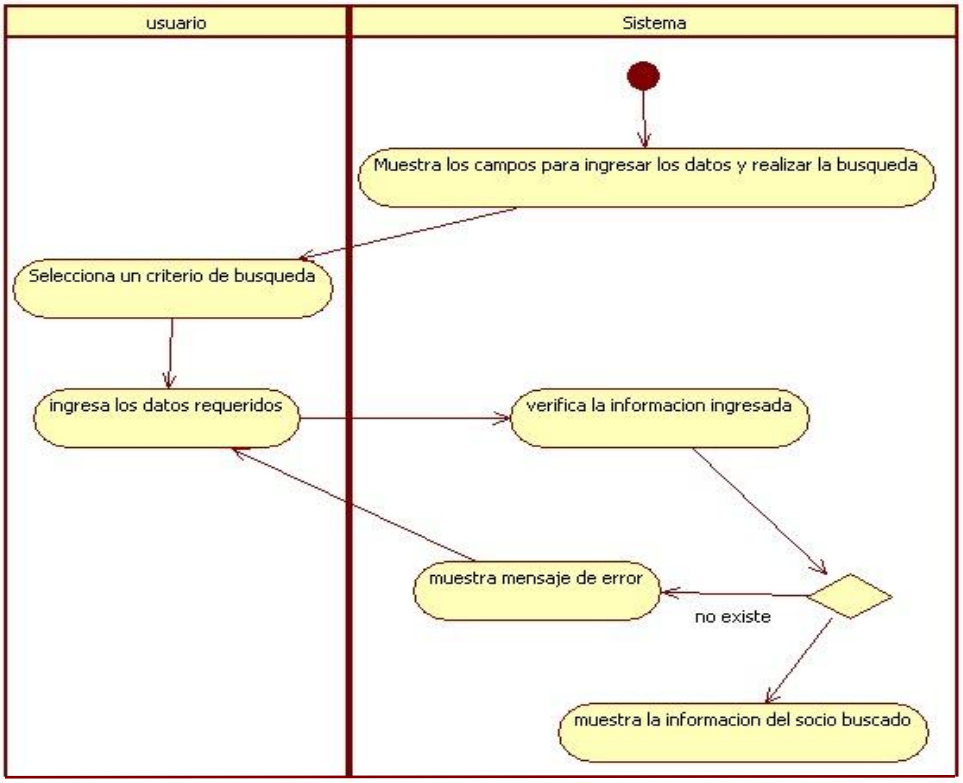
Dependencias	**FRQ-0001**
Descripción	La aplicación deberá permitir la eliminación de un usuario registrado en el sistema.
Importancia	Vital
Urgencia	Inmediata
Estado	Pendiente de verificación

4.2.3.5 Los escenarios son los siguientes:

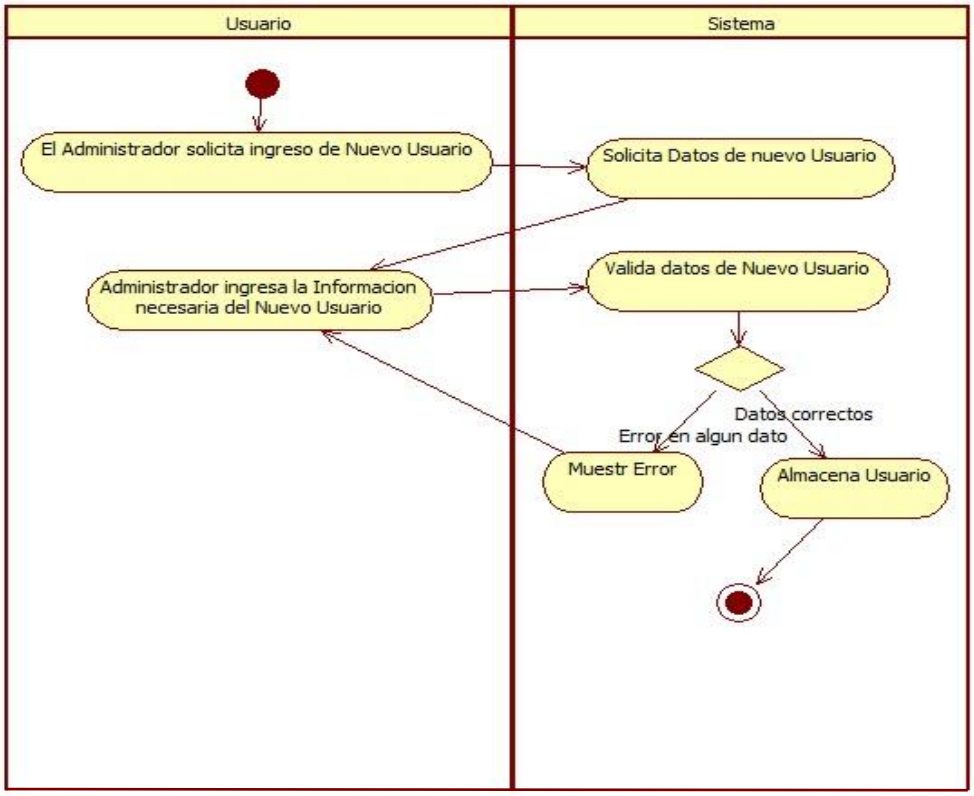
a) Ingresar al Sistema



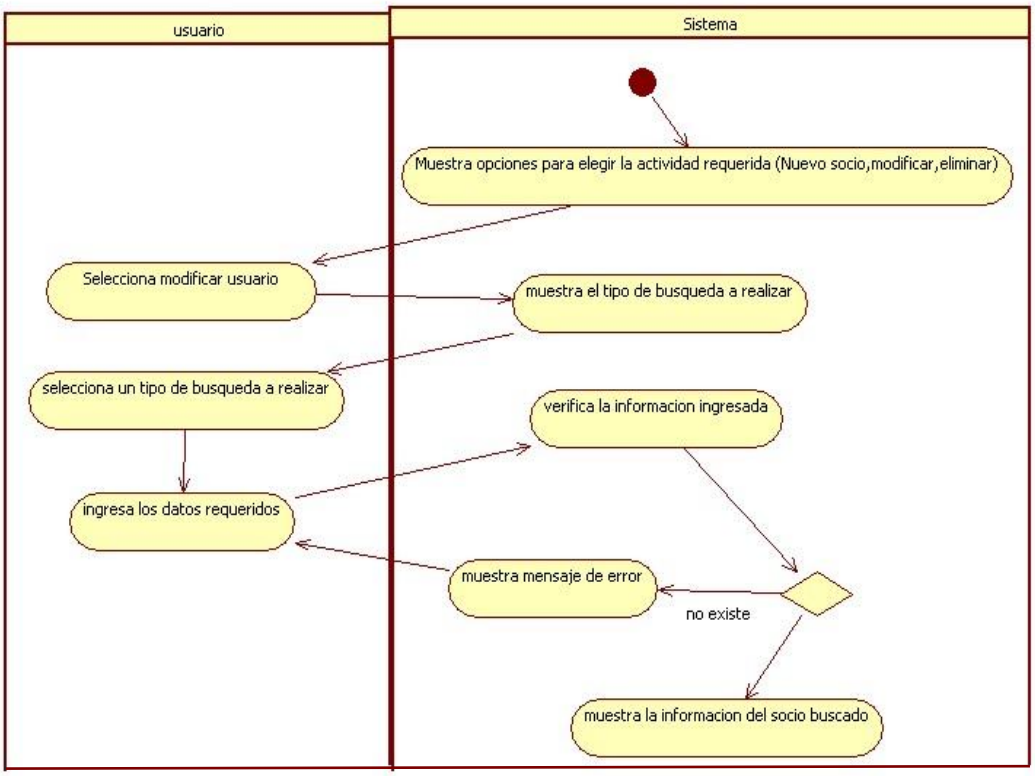
b) Búsqueda de Socio



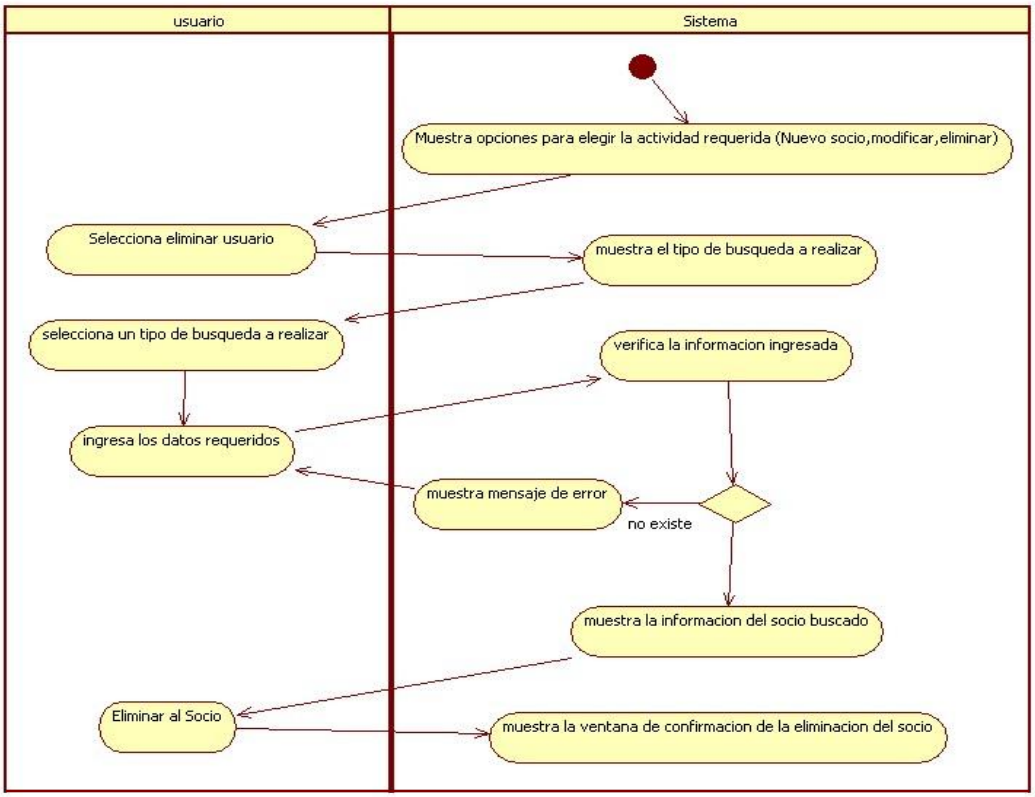
c) Registrar Socio



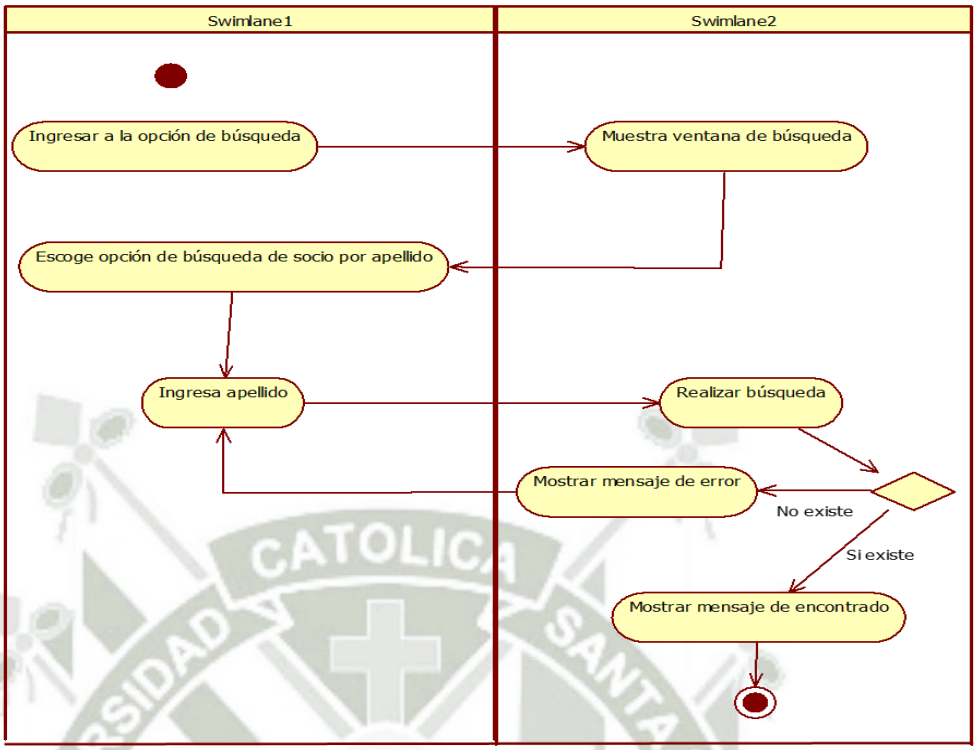
d) Modificar Socio



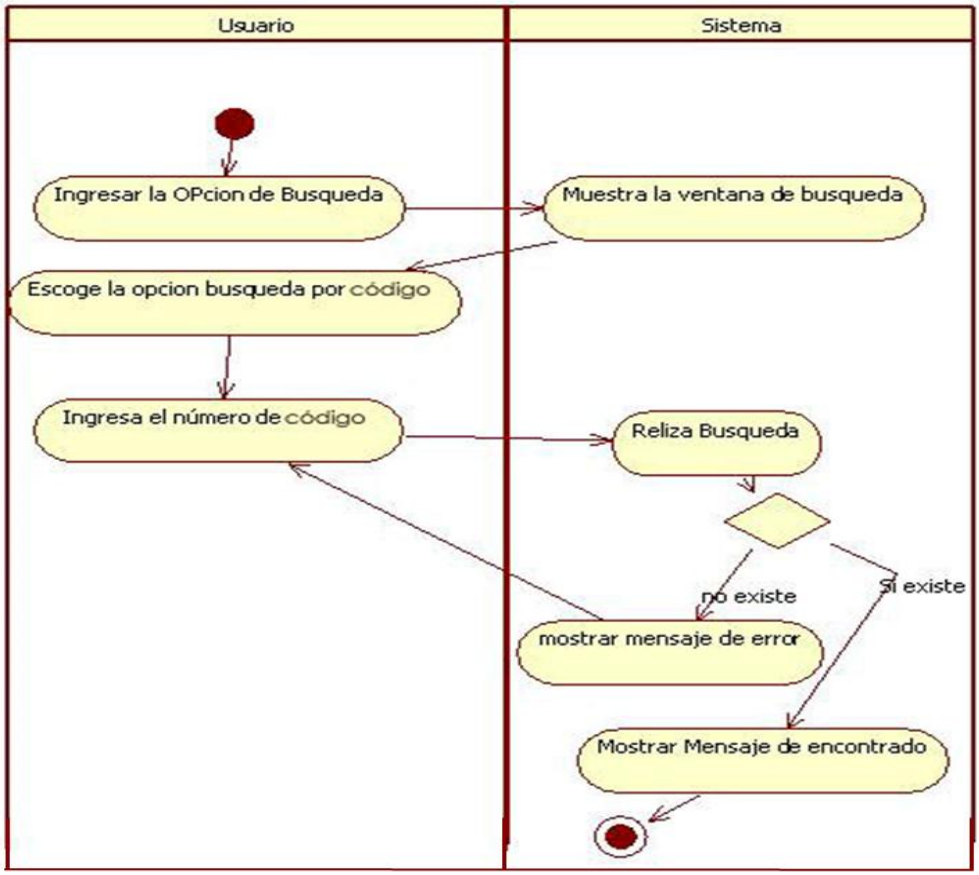
e) Eliminar Socio



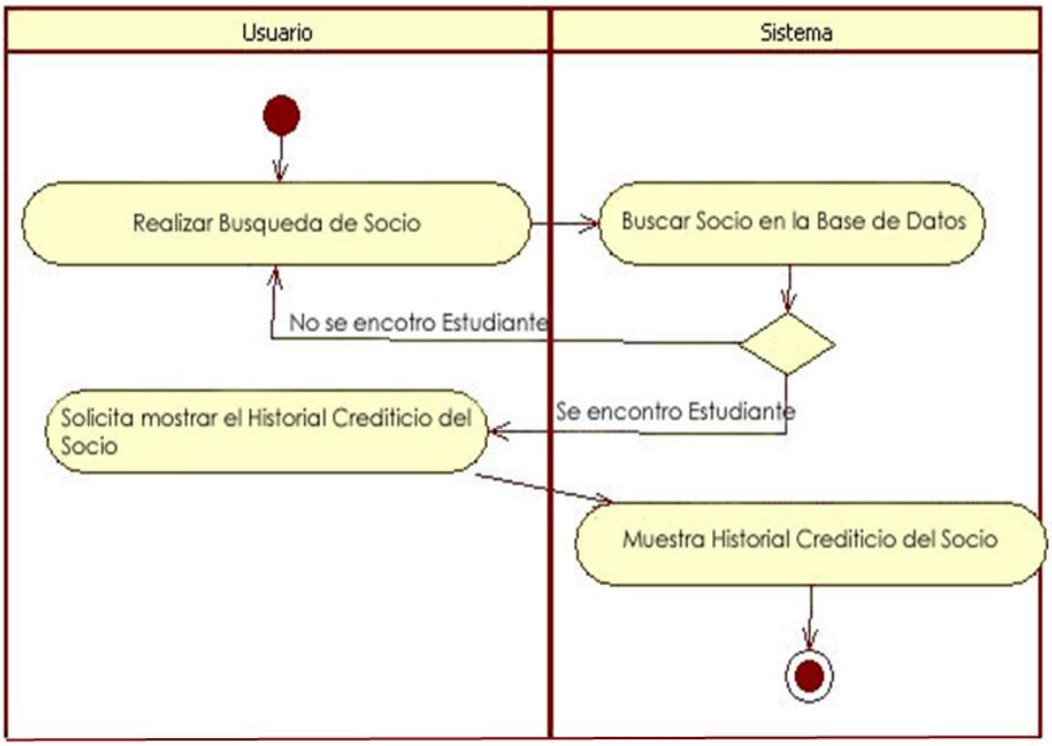
f) Buscar socio por apellido



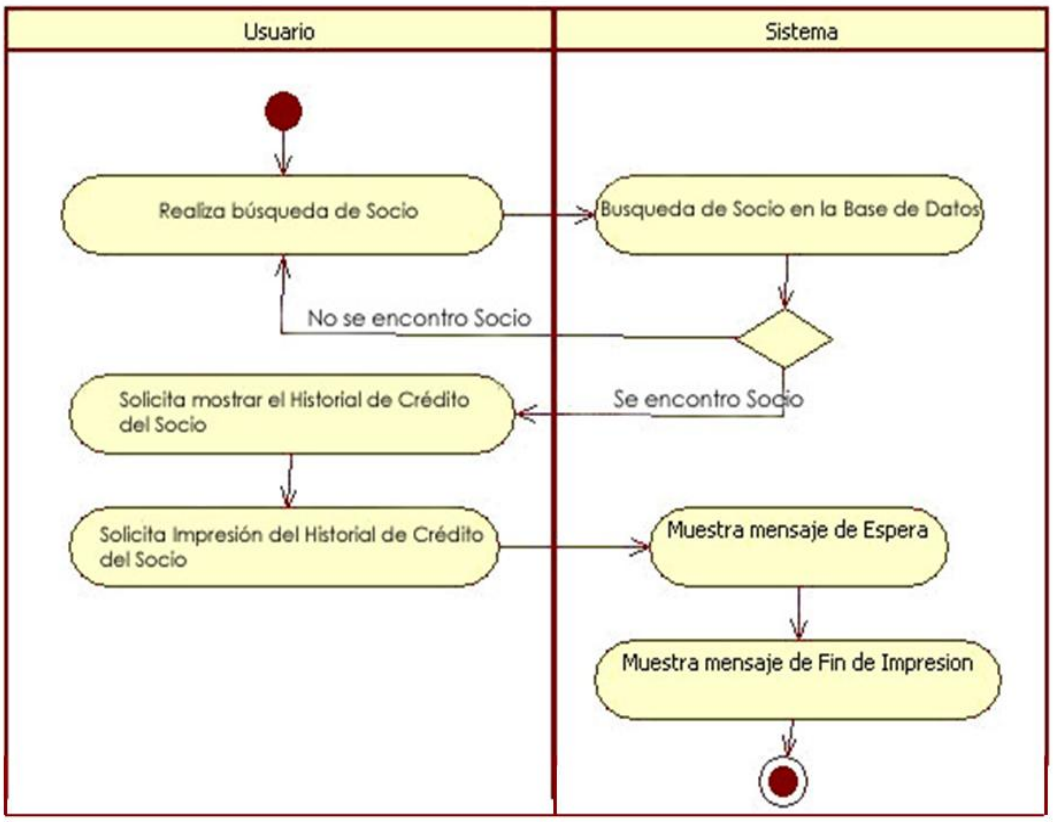
g) Buscar socio por código



h) Consultar Historial Crediticio

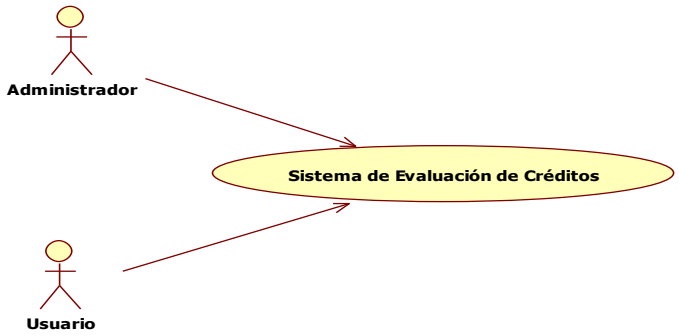


i) Imprimir Historial de Créditos



4.2.4 Casos de uso

4.2.4.1 Actores

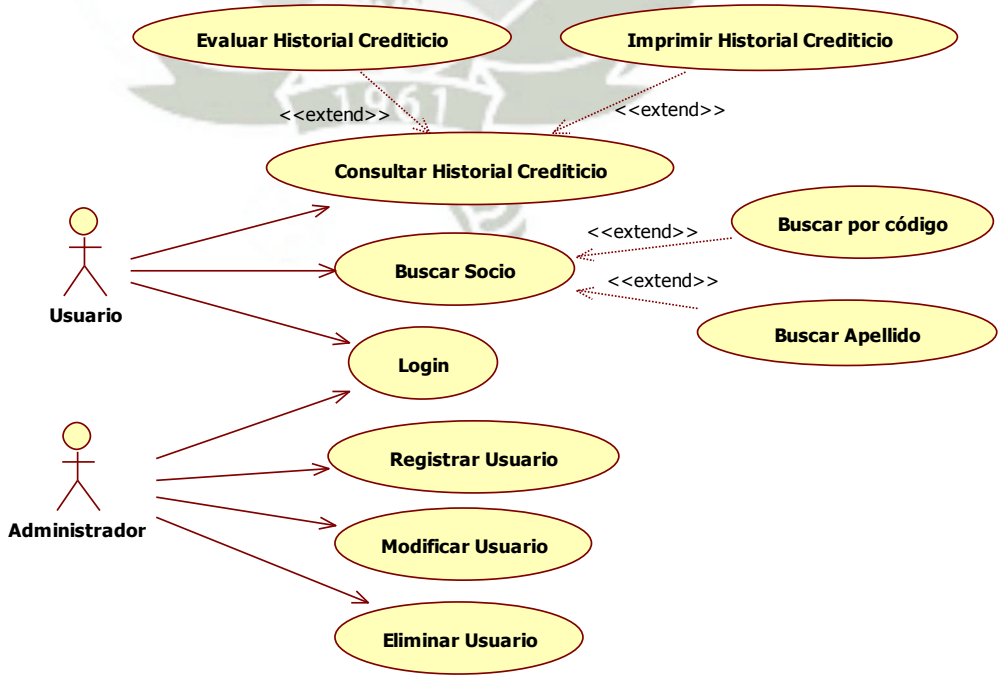


4.2.4.2 Especificación de Actores

ACT-0001	Usuario (Gestor y Jefe de Gestor)
Versión	0.1
Autores	Grupo de Analistas
Fuente	[STK-0001] [STK-0002] [STK-0003]
Descripción	Este actor hará uso del Sistema de Evaluación de Créditos.
Comentarios	Persona encargada [STK-0003]

ACT-0002	Administrador (encargado de sistemas)
Versión	0.1
Autores	Grupo de Analistas
Fuente	[STK-0001] [STK-0002]
Descripción	Este actor hará uso del Sistema de Evaluación de Créditos
Comentarios	Persona encargada [STK-0002] [STK-0002]

4.2.4.3 Diagrama de casos de uso reales



4.2.4.4 Descripción de Casos de Uso: Flujos Normales y Alternativos

a) Login

Nombre:	Login
Descripción:	Permite el ingreso del usuario o Administrador al Sistema
Referencias cruzadas:	CU-0001

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
	1. Muestra dos campos de llenado para completar con el nombre de ID y contraseña.
2. Ingresa su nombre de ID, contraseña y presiona en el botón ingresar.	3. Verifica que los datos ingresados estén registrados en la base de datos.
	4. El sistema muestra la ventana principal del Sistema de Evaluación de créditos.

Curso alternativo de eventos:

4. Si el nombre de ID y/o contraseña no son correctos entonces el sistema muestra un mensaje de error informando que no son correctos y que los vuelva a digitar.
5. Volver al paso 2.

b) Buscar Socio

Nombre:	Buscar Socio.
Descripción:	Permite la búsqueda de un Socio por código o apellido.
Referencias cruzadas:	CU-0002

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
	1. Muestra dos check y un campo de llenado para seleccionar el criterio de búsqueda e ingresar el código o apellido correspondiente del Socio.
2. Selecciona un criterio de búsqueda y luego ingresa el código o apellido y presiona en Buscar.	3. El sistema verifica la información ingresada.
	4. El sistema muestra la ventana con información del Socio buscado.

Curso alternativo de eventos 1:

4. Muestra mensaje de código no válido.
5. Volver al paso 2.

Curso alternativo de eventos 2:

4. Muestra mensaje de usuario no encontrado.
5. Volver al paso 2.

c) Registrar Usuario

Nombre:	Registrar Usuario.
Descripción:	Permite el registro de un nuevo Usuario.
Referencias cruzadas:	CU-0007

Curso normal de eventos:

Acción del Administrador	Respuesta del sistema
	1. Sistema muestra tres opciones Nuevo Usuario, Modificar Usuario, Eliminar Usuario.
2. Selecciona Nuevo Usuario	3. Muestra cuatro campos de texto correspondientes a usuario, contraseña, nombre y apellidos correspondientes al usuario.
4. Ingresa los datos solicitados, luego presiona guardar.	5. El sistema verifica la información ingresada.
	6. El sistema muestra la ventana confirmando que se ha guardado los datos del usuario exitosamente.

Curso alternativo de eventos:

6. Muestra mensaje de código no válido.
7. Volver al paso 3.

d) Modificar Usuario

Nombre:	Modificar Usuario
Descripción:	Permite realizar la modificación de un usuario
Referencias cruzadas:	CU-0008

Curso normal de eventos:

Acción del Administrador	Respuesta del sistema
	1. Sistema muestra tres opciones Nuevo Usuario, Modificar Usuario, Eliminar Usuario.
2. Selecciona Modificar Usuario	3. El sistema muestra la información de la lista de usuarios registrados.
4. El Administrador selecciona el usuario realiza la modificación correspondiente y presiona en guardar	5. El sistema verifica que la información modificada sea válida.
	6. El sistema muestra la ventana con datos modificados confirmando que se ha modificado los datos del usuario exitosamente.

Curso alternativo de eventos:

6. Muestra mensaje de Información no válida.
7. Volver al paso 3.

e) Eliminar Usuario

Nombre:	Eliminar Usuario
Descripción:	Permite realizar la eliminación de un usuario
Referencias cruzadas:	CU-0009

Curso normal de eventos:

Acción del Administrador	Respuesta del sistema
	1. Sistema muestra tres opciones Nuevo Usuario, Modificar Usuario, Eliminar Usuario.
2. Selecciona Eliminar Usuario	3. El sistema muestra la información de la lista de usuarios registrados.
4. El Administrador selecciona el usuario realiza la eliminación correspondiente y presiona en guardar	5. El sistema muestra la lista de usuarios registrados menos el que fue eliminado confirmando así que se ha eliminado al usuario exitosamente.

f) Buscar socio por Apellido

Nombre:	Buscar socio por apellido.
Descripción:	Permite realizar la búsqueda del socio por apellido.
Referencias cruzadas:	CU-0002

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
1. Realiza la búsqueda de un socio por Apellido y presionar en el botón buscar.	2. Muestra una lista con los siguientes datos: código del socio, nombre del socio, apellido del socio, en donde el nombre del socio puede coincidir.

g) Buscar socio por Código

Nombre:	Buscar socio por código.
Descripción:	Permite realizar la búsqueda del socio por código.
Referencias cruzadas:	CU-0003

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
1. Realiza la búsqueda de un socio por código y presionar en el botón buscar.	2. El sistema muestra varios campos de llenado en donde muestra el código, nombre, nivel, interés, monto, viabilidad, numero de cuotas. También muestra una lista con los siguientes datos: Número, fecha, amortización, interés, total y mora en días.

h) Consultar Historial Crediticio

Nombre:	Consultar Historial Crediticio
Descripción:	Permite realizar la consulta del Historial Crediticio.
Referencias cruzadas:	CU-0004

Curso normal de Eventos:

Acción del Usuario	Respuesta del sistema
	1. El sistema muestra la información del criterio de búsqueda de usuarios por código o apellido
2. Ingresa los datos correspondientes dependiendo del criterio de Búsqueda	3. El sistema verifica la información ingresada
	4. El sistema muestra la ventana con información del socio buscado.

i) Imprimir Historial Crediticio

Nombre:	Imprimir Historial Crediticio
Descripción:	Permite imprimir el Historial Crediticio de un socio.
Referencias cruzadas:	CU-0006

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
	1. El sistema muestra la ventana con información del socio buscado.
2. El usuario presiona en imprimir.	3. El sistema muestra un mensaje de transacción exitosa.

j) Evaluar Historial Crediticio

Nombre:	Evaluar Historial Crediticio
Descripción:	Permite evaluar el Historial Crediticio de un socio.
Referencias cruzadas:	CU-0005

Curso normal de eventos:

Acción del usuario	Respuesta del sistema
	1. El sistema muestra la ventana con información del socio buscado.
2. El usuario realiza la evaluación correspondiente del socio.	3. El sistema muestra un mensaje de transacción exitosa.

4.2.5 Calidad del producto

DOCUMENTOS	NTP
1. METODOLOGÍA DE DESARROLLO DE SOFTWARE	x
2. PLAN DE DESARROLLO DE SOFTWARE	x
3. REGLAMENTO INTERNO	x
4. DOCUMENTO DE CONTRATO	x
5. EDUCCION DE REQUERIMIENTOS	x
6. ELICITACION DE REQUERIMIENTOS	x
7. REQUERIMIENTOS FUNCIONALES	x
8. REQUERIMIENTOS NO FUNCIONALES	x
9. DETALLE DE CASOS REALES DE USO	x
10. DOCUMENTO DE DIAGRAMA DE CLASES	x
11. DIAGRAMA DE ACTIVIDADES	x
12. DIAGRAMA DE SECUENCIA	x
13. DIAGRAMA DE COLABORACION	x
14. DOCUMENTO DE INTERFACES	x
15. DEFINICION DE BASE DE DATOS	x
16. DOCUMENTO DE DISEÑO DE ARQUITECTURA	x
17. DOCUMENTO DE ESTABLECIMIENTO DE CALIDAD	x
18. DOCUMENTO DE MATRIZ DE TRAZABILIDAD	x
19. DOCUMENTO DE PRUEBAS	x
20. MANUAL DE USUARIO	x

Matriz de Documentos por la Norma Técnica Peruana

a) Matriz de trazabilidad

MATRIZ DE CASOS REALES DE USO X DIAGRAMA DE SECUENCIA

CASOS DE USO REALES/DS	DS001	DS002	DS003	DS004	DS005	DS006	DS007	DS008	DS009	DS010
1.	X									
2.				X						
3.							X			
4.								X		
5.									X	
6.						X				
7.					X					
8.										X
9.			X							
10.		X								

b) Lista de Casos de Uso Reales

Se hace referencia a los casos reales de uso del documento **"Detalles de casos de uso"**

1. Login.

2. Buscar Socio

3. Registrar Usuario

4. Modificar Usuario

5. Eliminar Usuario

6. Buscar socio por Apellido

7. Buscar socio por Código

8. Consultar Historial Crediticio

9. Imprimir Historial Crediticio

10. Evaluar Historial Crediticio

c) Lista de Diagramas de secuencia

Se hace referencia a los casos reales de uso del documento **"Documento diagrama de secuencia".**

DS001 Login

DS002 Consultar Historial Crediticio

DS003 Imprimir Historial

DS004 Buscar Socio

DS005 Buscar por Código

DS006 Buscar por Apellido

DS007 Registrar Usuario

DS008 Modificar Usuario

DS009 Eliminar Usuario

DS010 Evaluar Historial Crediticio

4.2.6 Pruebas de software

Resumen

Este documento nos proporciona una vista general para desarrollar las distintas pruebas utilizando HttpUnit como herramienta para este sistema.

a) Pruebas Unitarias

La prueba unitaria se encarga de probar el correcto funcionamiento de un módulo. Asegurando que cada uno de los módulos funcione correctamente por separado. El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas.

Debemos escribir casos de prueba para cada función no trivial o método en sus respectivos módulos de forma que cada caso sea independiente del resto**.**

Características

- Automatizable: Es útil para integración continúa.
- Repetibles o Reutilizables: Se deben crear pruebas que se pueden ejecutar varias veces.
- Completas: Abarcan la mayor cantidad de código.
- Profesionales: Las pruebas deben ser consideradas con la misma profesionalidad que el código, documentación, etc.
- Independientes: La ejecución de una prueba no debe afectar a la ejecución de otra.

Ventajas

Estas pruebas proporcionan cinco ventajas básicas:

- Los errores están más acotados facilitando su localización.
- Simplifica la integración: Ya que se quiere llegar a la fase de integración con un grado de seguridad muy elevada.
- Separación de la interfaz y la implementación: La única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.
- Documenta el código: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
- Fomentan el cambio: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura, puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han producido errores.

Limitaciones

Las pruebas unitarias no descubrirán todos los errores del código sólo prueban las unidades por sí solas. Por lo tanto, no descubrirán errores de integración, problemas de rendimiento y otros problemas que afectan a todo el sistema en su conjunto. Las pruebas unitarias sólo son efectivas si se usan en conjunto con otras pruebas de software.

Herramientas

Entre las herramientas que se desarrollarán con más detalle se mencionan las siguientes:

- HttpUnit
- HtmlUnit
- JwebUnit

Al finalizar la construcción del software, se practican los siguientes casos de prueba para saber que el sistema cumple con los requisitos planteados en un inicio por el cliente (stakeholder).

	Correcto	Desambiguo	Completo	Consistente	Ranking por importancia y/o estabilidad	Verificable	Modificable	Trazable
Login	No	No	No	Si	No	Si	Si	Si
Buscar Socio	No	No	No	Si	No	Si	Si	Si
Registrar Usuario	No	No	Si	Si	Si	Si	Si	Si
Modificar Usuario	No	No	Si	Si	So	Si	Si	Si
Eliminar Usuario	No	No	No	Si	No	Si	Si	Si
Buscar socio por Apellido	No	No	Si	Si	Si	Si	Si	Si
Buscar socio por Código	No	No	Si	Si	Si	Si	Si	Si
Consultar Historial Crediticio	No	No	Si	Si	No	Si	Si	Si

Imprimir Historial Crediticio	No	No	Si	Si	No	Si	Si	Si
Evaluar Historial Crediticio	No	No	No	Si	Si	Si	Si	Si
Consultar avales internos	No	No	No	Si	No	Si	Si	Si
Consultar avales externos	No	No	Si	Si	Si	Si	Si	Si
Consultar créditos externos	No	No	Si	Si	Si	Si	Si	Si
Consultar central de riesgo	No	No	Si	Si	Si	Si	Si	Si
Consultar pagos externos	No	No	Si	Si	Si	Si	Si	Si
Consultar boletas de pago	No	No	Si	Si	Si	Si	Si	Si
Consultar otras boletas de pago	No	No	No	Si	No	Si	Si	Si
Deducir expectativas de pago	No	No	No	Si	No	Si	Si	Si
Deducir pagos fijos	No	No	Si	Si	Si	Si	Si	Si
Deducir pagos variables	No	No	Si	Si	Si	Si	Si	Si
Consolidar pagos	No	No	No	Si	No	Si	Si	Si
Consolidar deudas	No	No	No	Si	No	Si	Si	Si
Consolidar avales	No	No	No	Si	No	Si	Si	Si

b) Herramientas de Pruebas

Las herramientas de pruebas son aquel conjunto de programas o bibliotecas que nos permiten realizar las pruebas para las aplicaciones de manera automática. Usando este tipo de programas podemos descubrir los errores existentes en nuestro código para proceder a la corrección de los mismos. En este campo, podemos dividir las herramientas existentes en dos tipos: en primer lugar aquellos productos software que son independientes del entorno de desarrollo y en segundo lugar aquellas bibliotecas que se pueden importar al entorno de desarrollo para realizar las pruebas.

Para realizar este pequeño estudio sobre herramientas de pruebas nos inclinaremos a escoger aquellas que se corresponden con bibliotecas para diferentes plataformas, ya que estas pueden ser utilizadas desde diversos entornos de desarrollo de manera sencilla.

Httpunit

Es una extensión de Junit para la realización de pruebas unitarias sobre el protocolo http.

- Utiliza la metáfora de conversación con el servidor web, al estilo de htmlunit.
- Permite obtener el texto de la página o su estructura DOM, reconociendo sus principales elementos (tablas, formularios, frames. etc.)
- La documentación del framework es escasa
- No indican el grado de soporte de funciones javascript

Htmlunit

Es una extensión de Junit para la realización de pruebas unitarias web. Su funcionamiento se basa en obtener los diferentes elementos que componen la página web devuelta por el servidor, e interactuar sobre ellos. Algunas de sus características son:

- Objetos para todas las etiquetas html: title, table, etc.
- Gestión de los eventos javascript, como onclick, etc. Esto todavía está en desarrollo.

- Sintaxis bastante farragosa. Para acceder a los elementos de la página hay que ir instanciando los diferentes objetos que la soportan.
- Gestión parcial de los eventos javascript de los elementos html de la página. Todavía no está completo el soporte, aunque sí dispara el onclick()
- El proyecto está bastante activo, y se ha liberado en 2006 la versión 1.10

Jwebunit

Es un framework de pruebas unitarias web basado en Htmlunit, que simplifica el desarrollo de las pruebas. Dispone de las siguientes características:

- Sintaxis sencilla para las pruebas, mucho más simples que en Htmlunit.
- Posibilidad de usar diferentes plugins para acceder a la página web a probar. La nueva versión en preparación (2.0) soportará cuatro plugins diferentes: htmlunit, httpunit, Selenium y Jwebfit
- Rapidez a la hora de hacer las pruebas unitarias

c) Herramienta seleccionada y justificación de la decisión

Debido a que la realización que la aplicación será web, las pruebas unitarias de la aplicación se desarrollarán utilizaríamos la herramienta HtmlUnit.

Si tenemos que realizar un conjunto de pruebas unitarias en nuestra aplicación, las dos partes fundamentales a probar serian:

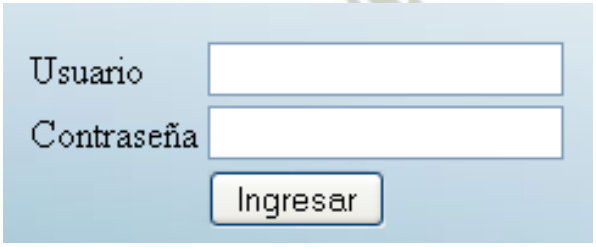
- La interfaz de la aplicación, incluyendo la navegación. Debemos comprobar que la aplicación cumple los casos de uso que hemos determinado, realizando la navegación correcta por las diferentes páginas necesarias, y que muestra en todo momento la información correcta al usuario.
- La lógica de la aplicación. Una vez que la aplicación se comporta externamente de manera correcta, es decir que se comporte correctamente. Es el momento de probar la lógica de la aplicación, los diferentes estados y la persistencia en base de datos, como elementos frecuentes.

De entre todos los productos citados, nuestros candidatos para realizar las pruebas unitarias serían:

- Para las pruebas de interfaz, un buen candidato es Jwebunits. Con este framework podremos simular el comportamiento de los usuarios con la aplicación de modo sencillo.

d) Verificación por medio de los interfaces de usuario

d.1. Interfaz de Login

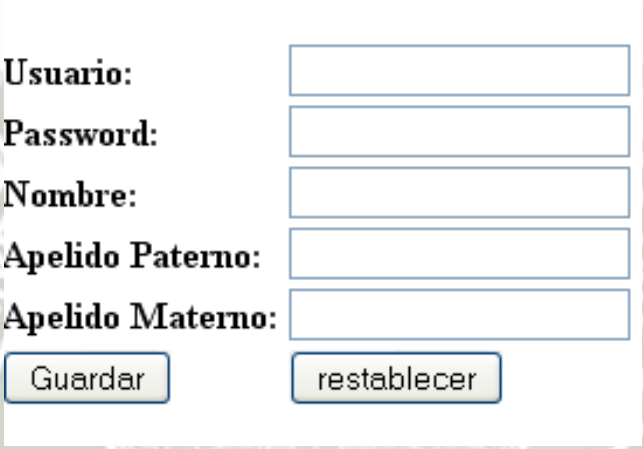


Pantalla Número		Nombre de la Pantalla		ID Formulario
1		Login		login
Versión	0.6	**Referencia**		CU-0001
Descripción	Nos permite ingresar al sistema, para ello se debe estar registrado como usuario.			

REF	Tipo	Nombre del Campo	Mensaje de Error	Mensaje de Ayuda
1	Campo de texto	txtUsuario		
2	Campo de Texto	txtPass	Password incorrecto	

BOTON	ACCIÓN	BOTON	ACCIÓN
Ingresar	Ingresar al Sistema		

d.2. Interfaz de Nuevo Usuario

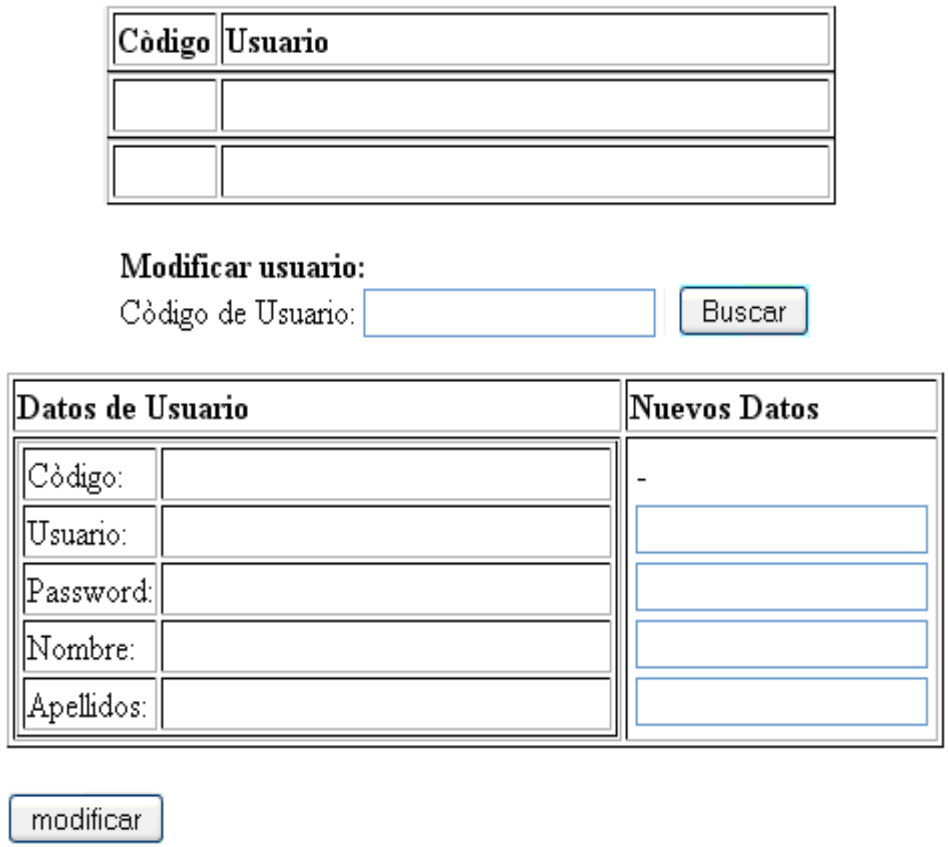


Pantalla Número		Nombre de la Pantalla		ID Formulario
2		Nuevo Usuario		Nuevo Usuario
Versión	0.6	**Referencia**		CU-0007
Descripción	Nos permite ingresar al sistema nuevos usuarios, se debe estar registrado como administrador para ingresar a esta interfaz.			

REF	Tipo	Nombre del Campo	Mensaje de Error	Mensaje de Ayuda
1	Campo de texto	Txtusunuevo		
2	Campo de Texto	Txtpasnuevo		
3	Campo de Texto	Txtnomuevo		
4	Campo de Texto	Txtappatnuevo		
5	Campo de Texto	Txtapmatnuevo		

BOTON	ACCIÓN	BOTON	ACCIÓN
Guardar	Ingresar al Sistema	restablecer	Limpia el contenido de las cajas de texto

d.3. Interfaz de Modificar Usuario

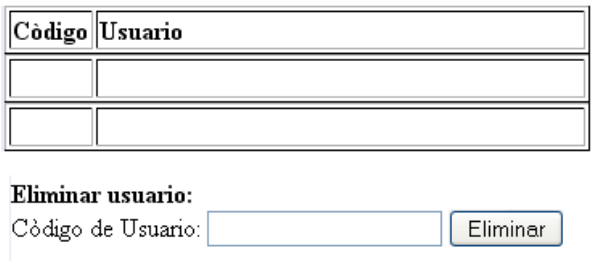


Pantalla Número		Nombre de la Pantalla		ID Formulario
3		Modificar Usuario		ModUsuario
Versión		0.6	**Referencia**	CU-0008
Descripción	Nos permite ingresar modificar datos de los usuarios del sistema, para ello se debe estar registrado como usuario Administrador.			

REF	Tipo	Nombre del Campo	Mensaje de Error	Mensaje de Ayuda
1	Campo de texto	Txtcodusuario	Codigo no existe	
2	Tabla	Tabvisusuario		
3	Tabla	Tabmodusuario		

BOTON	ACCIÓN	BOTON	ACCIÓN
modificar	Actualiza los cambios realizados en los datos del usuario	Buscar	Busca el código ingresado y lo muestra en la tabla tabmodusuario.

d.4. Interfaz de Eliminar Usuario



Pantalla Número		Nombre de la Pantalla		ID Formulario
4		Eliminar Usuario		Eliminar Usuario
Versión	0.6	**Referencia**		CU-0009
Descripción	Nos permite eliminar un usuario seleccionado para ello se debe estar registrado como usuario Administrador			

REF	Tipo	Nombre del Campo	Mensaje de Error	Mensaje de Ayuda
1	Campo de texto	Txtelimsuario	Código no existe	
2	Tabla	Tabviselimusuario		

BOTON	ACCIÓN	BOTON	ACCIÓN
Eliminar	Elimina al usuario seleccionado del sistema		

d.5. Interfaz de Búsqueda

Pantalla Número	Nombre de la Pantalla		ID Formulario
5	Búsqueda		Búsqueda
Versión	0.6	**Referencia**	CU-0002,CU-0003
Descripción	Nos permite realizar búsquedas de socios que han tenido créditos para acceder debe estar registrado como usuario.		

REF	Tipo	Nombre del Campo	Mensaje de Error	Mensaje de Ayuda
1	Campo de Texto	txtDatos	Ingrese dato	No existen datos de búsqueda
2	Campo de Texto	Rbcod	Ingrese dato	No existen datos de búsqueda.

BOTON	ACCIÓN	BOTON	ACCIÓN
Buscar	Realiza búsqueda de acuerdo a la selección realizada en los campos de texto por código (Nos lleva a la interfaz de Reporte de crédito) y por apellido(Nos lleva a la interfaz de listado de clientes)		

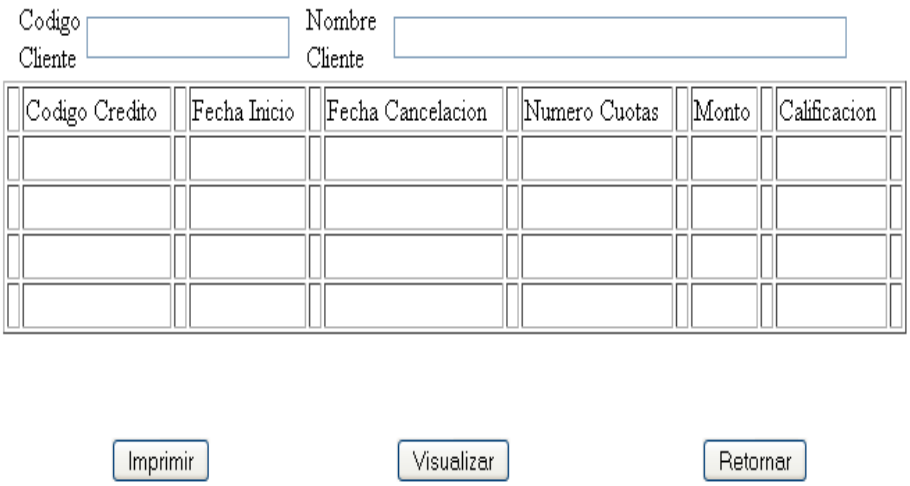
d.6. Interfaz Listado de Clientes

Pantalla Número		Nombre de la Pantalla		ID Formulario
6		Listado de Clientes		BporApellido
Versión		0.6	**Referencia**	CU-006
Descripción	Nos permite visualizar un listado de clientes por apellido a través de una tabla			

REF	Tipo	Nombre del Campo	Mensaje de Error	Mensaje de Ayuda
1	Tabla	Tbusape	Seleccionar un cliente	

BOTON	ACCIÓN	BOTON	ACCIÓN
Atras	Nos permite retornar a la interfaz de búsquedas para realizar otra búsqueda	Visualizar	Nos permite acceder a la interfaz de Reporte de crédito si se ha seleccionado un determinado cliente en la tabla de listado de clientes.

d.7. Interfaz de Reporte de Crédito



Pantalla Número		Nombre de la Pantalla		ID Formulario
7		Reporte de Crédito General		RepGeneral
Versión	0.6		**Referencia**	CU-0004
Descripción	Nos permite visualizar el reporte de todos los créditos realizados de un determinado cliente previamente seleccionado.			

REF	Tipo	Nombre del Campo	Mensaje de Error	Mensaje de Ayuda
1	Tabla	Trgrepgeneral		
2	Campo de texto	Txtrgcodcliente		
3	Campo de texto	Txtrgcodcliente		

BOTON	ACCIÓN	BOTON	ACCIÓN
Retornar	Nos permite retornar a la interfaz de busquedas para realizar otra busqueda	Visualizar	Nos permite acceder a la interfaz de Reporte de crédito si se ha seleccionado un determinado cliente en la tabla de de listado de clientes.
Imprimir	Nos permite imprimir el reporte detallado del crédito visualizado		

5.1 Evaluación por expertos

Para evaluar la técnica propuesta se tomó la decisión de requerir a dos tipos de evaluadores, a aquellos que tienen experiencia en el desarrollo de software y a los usuarios. El juicio emitido por los expertos permitirá determinar la implementación de pruebas de software a realizar, es así que para la selección de evaluadores se tomó el criterio de elegir a aquellos con los conocimientos necesarios.

5.2 Perfil del experto

Los expertos son personas que tienen la característica de tener muy buenos conocimiento acerca de la forma como se deben de implementar pruebas de software ya que han participado en proyectos donde se les asignaba esta tarea. Los expertos son personas ajenas al proyecto y a la organización donde se implementarán estas pruebas.

5.3 Ponderaciones y evaluación

Para llevar a cabo la evaluación de la técnica, en primer lugar se propuso la valoración de las puntuaciones con respecto a los desarrolladores expertos y aquellos desarrolladores inexpertos que tienen a su cargo la evaluación del producto de software. Las valoraciones propuestas fueron sugeridas por los expertos. La tabla 3 muestra las ponderaciones de acuerdo al nivel de satisfacción del usuario.

Tabla 3
Valoración según nivel de satisfacción

Nemotécnico	Descripción	Valoración	
		Expertos	Inexpertos
A	Muy bueno	8	4
B	Bueno	6	3
C	Regular	3	2
D	Malo	1	1
E	Pésimo	0	0

Fuente: Elaborado por el autor

De igual modo quedan definidos los valores, de acuerdo a los vistos en la tabla 4, de cada uno de los parámetros a tomar en cuenta para llevar a cabo la evaluación de las pruebas de software a implementar. La tabla 4 muestra la distribución de los valores llevados a cabo.

Tabla 4
Valoración según parámetros

Parámetro	Expertos					Inexpertos				
	A	B	C	D	E	A	B	C	D	E
Confiabilidad	8	6	3	1	0	4	3	2	1	0
Aceptabilidad	8	6	3	1	0	4	3	2	1	0
Operatividad	6	4	2	1	0	4	3	2	1	0

Fuente: Elaborado por el autor

Para llevar a cabo la valoración de la técnica, se trabajó sobre el módulo "Evaluación de Créditos" elaborado por la empresa desarrolladora de software Objetive Software. A todos los participantes se les hizo una explicación detallada de la técnica y el objetivo que perseguía. Después, y por separado, se les entregaba las pruebas de software implementadas para su evaluación. La tabla 5 muestra los resultados de la evaluación.

Se escogieron a tres desarrolladores expertos en la implementación de pruebas de software y seis desarrolladores inexpertos. Los desarrolladores inexpertos fueron seleccionados por el constructor del producto.

Tabla 5
Comparación de técnicas o métodos

Parámetro	Expertos				Usuarios						
	Exp1	Exp2	Exp3	PExp	Inex1	Inex2	Inex3	Inex4	Inex5	Inex6	PInex
Confiabilidad	6	6	6	6.00	3	2	2	3	3	2	2.00
Aceptabilidad	4	6	4	4.67	3	3	3	2	2	2	2.00
Operatividad	4	4	6	4.67	3	2	3	3	2	3	2.17

Fuente: Elaborado por los autores

De la calificación llevada a cabo por los desarrolladores expertos se puede concluir que las pruebas de software propuestas tienen una buena aceptación para los parámetros de confiabilidad (el puntaje máximo es de 8 puntos). De igual manera en la aceptabilidad y operatividad el puntaje obtenido se acerca al límite de la escala (6 puntos), lo que implica que para los expertos las pruebas implementadas son las adecuadas.

Con respecto a los desarrolladores inexpertos, todos los parámetros obtienen puntajes que se acercan límite superior (3 puntos) lo que significa que las pruebas de software implementadas son medianamente adecuadas.

Con la finalidad de medir el efecto que causa cada uno de los evaluadores, se emplea la relación vista en [HERTZUM, 1999], quien menciona que para medir este efecto se debe de aplicar la siguiente relación:

$$Tasa = promedio\ de\ (\frac{Pi}{Ptodos})$$

En donde Pi es el conjunto de problemas detectados por el evaluador y Ptodos es el conjunto de problemas detectados por todos los evaluadores. La Tabla 6 muestra La tasa de detección de problemas.

Tabla 6
Tasa de detección de problemas

Parámetro	Expertos				Usuarios						
	Exp1	Exp2	Exp3	Total	Inex1	Inex 2	Inex3	Inex 4	Inex5	Inex6	Total
Confiabilidad	12	8	10	30.00	8	7	6	7	5	5	38.00
Aceptabilidad	7	11	8	26.00	5	6	7	4	3	6	31.00
Operatividad	13	7	7	27.00	4	6	5	3	6	7	31.00
TOTAL	32	26	25	83.00	17	19	18	14	14	18	100.00
Pi	0.39	0.31	0.30	1.00	0.17	0.19	0.18	0.14	0.14	0.18	1.00
Promedio			0.33					0.17			

Fuente: Elaborado por el autor

La tabla 6 muestra que los expertos presentan una tasa de detección de problemas de alrededor del 33% mientras que los desarrolladores inexpertos tienen una tasa promedio de detección de problemas del 17%. Estos valores corroboran las conclusiones obtenidas durante la primera evaluación. La evaluación se lleva a cabo independientemente del proceso de construcción de software.

5.4 Resultados de la encuesta

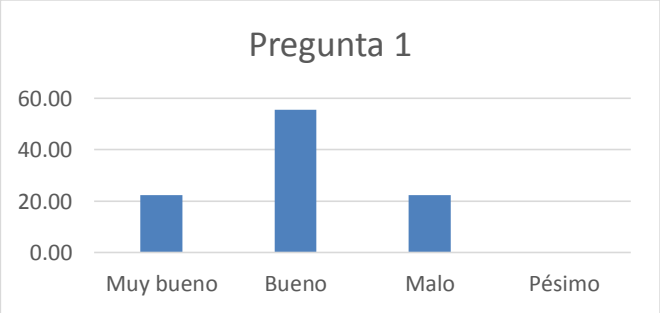
Para obtener una opinión sobre la técnica propuesta, se aplicó la encuesta descrita en el Anexo N° 2 a las nueve personas descritas en el numeral anterior (desarrolladores expertos e inexpertos). Los resultados se muestran a continuación:

PREGUNTA 1: ¿Cómo califica la técnica para detectar las causas a los problemas de las pruebas de software?

Tabla 7
Pregunta 1

CARACTERISTICA	CANTIDAD	%
Muy bueno	2	22.22
Bueno	5	55.56
Malo	2	22.22
Pésimo	0	0.00
TOTAL	**9**	**100.00**

Fuente: Elaborado por el autor



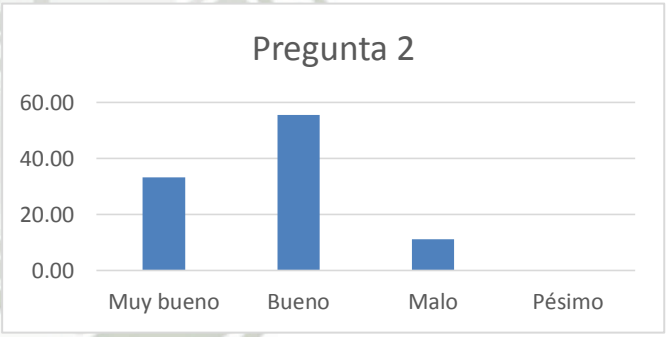
El 22.22% manifiestan que la técnica es muy buena cuando se trata de detectar las causas a los problemas de las pruebas de software. Asimismo el 55.56% manifiesta que la técnica es buena. Sólo el 22.22% de los encuestados no encontraron a la técnica adecuada. En conclusión el 77.78% encontraron que la técnica es buena para detectar las causas de los problemas de las pruebas de software.

PREGUNTA 2: ¿Cómo califica la definición de las métricas para reconocer el estado de las prácticas en el ciclo de desarrollo de software?

Tabla8
Pregunta 2

CARACTERISTICA	CANTIDAD	%
Muy bueno	3	33.33
Bueno	5	55.56
Malo	1	11.11
Pésimo	0	0.00
TOTAL	**9**	**100.00**

Fuente: Elaborado por el autor



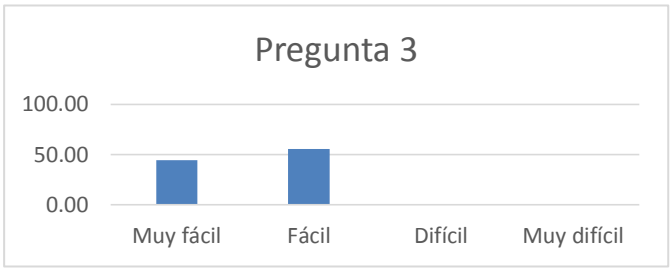
El 33.33% de los encuestados manifestaron que es muy buena la definición de métricas que permite reconocer el estado de las prácticas en el ciclo de desarrollo de software. Asimismo el 55.56% de los mismos indicaron que esta definición es buena y solo el 11.11% mencionaron que era malo. En conclusión el 88.89% manifestaron que la definición de métricas era buena.

PREGUNTA 3: ¿Cómo califica al proceso de soporte sugerido para la formulación de pruebas de software?

Tabla 9
Pregunta 3

CARACTERISTICA	CANTIDAD	%
Muy fácil	4	44.44
Fácil	5	55.56
Difícil	0	0.00
Muy difícil	0	0.00
TOTAL	**9**	**100.00**

Fuente: Elaborado por el autor



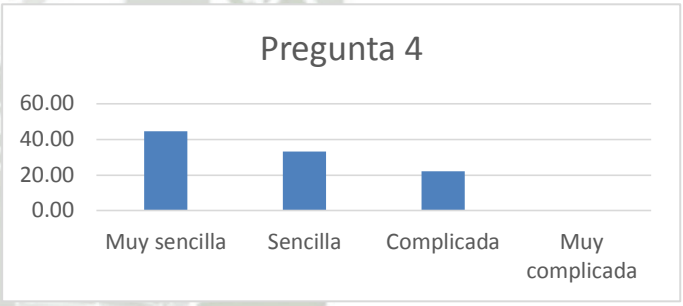
El 44.44% de los encuestados manifestaron que el proceso de soporte era muy fácil mientras que el 55.56% indicaron que era fácil. En conclusión, el 100% indicaron que el proceso de soporte es muy fácil.

PREGUNTA 4: ¿Cómo califica usted la técnica propuesta que permite implementar pruebas de software?

Tabla 10
Pregunta 4

CARACTERISTICA	CANTIDAD	%
Muy sencilla	4	44.44
Sencilla	3	33.33
Complicada	2	22.22
Muy complicada	0	0.00
TOTAL	**9**	**100.00**

Fuente: Elaborado por el autor



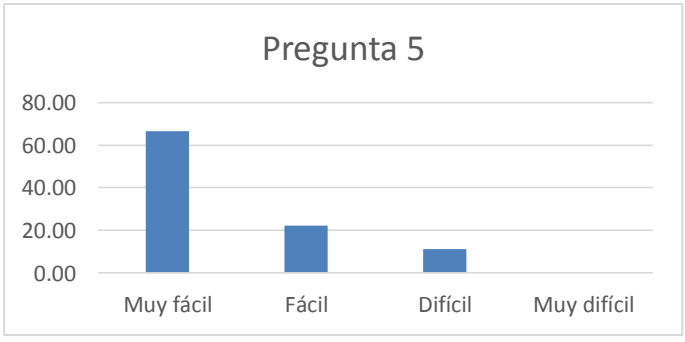
El 44.44% de los entrevistados indicaron que la técnica era muy sencilla de usar mientras que el 33.33% indicaron que era sencilla. Solo el 22.22% indicaron que la técnica era complicada. En conclusión, el 77.77% de los entrevistados indicaron que la técnica era sencilla para comprenderla.

PREGUNTA 5: ¿La técnica propuesta es fácil de comprender?

Tabla 11
Pregunta 5

CARACTERISTICA	CANTIDAD	%
Muy fácil	6	66.67
Fácil	2	22.22
Difícil	1	11.11
Muy difícil	0	0.00
TOTAL	**9**	**100.00**

Fuente: Elaborado por el autor



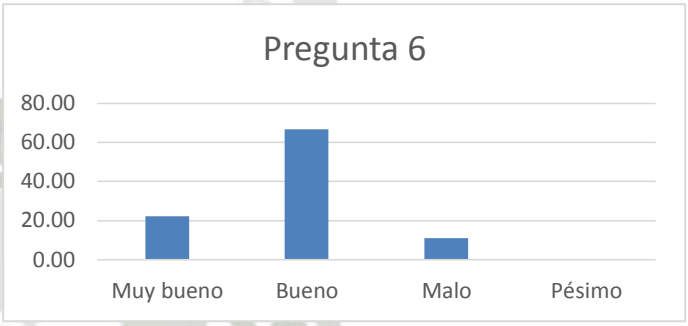
El 66.67% de los encuestados manifestaron que la técnica era muy fácil de comprender mientras que el 22.22% indicaron que era fácil. El 11.11% manifestaron que la técnica era difícil. En conclusión el 88.89% de los entrevistados que la técnica era sencilla de comprender.

PREGUNTA 6: ¿Cómo califica usted la forma de escalamiento de la técnica que permite detectar los problemas en las pruebas de software?

Tabla 12
Pregunta 6

CARACTERISTICA	CANTIDAD	%
Muy bueno	2	22.22
Bueno	6	66.67
Malo	1	11.11
Pésimo	0	0.00
TOTAL	**9**	**100.00**

Fuente: Elaborado por el autor



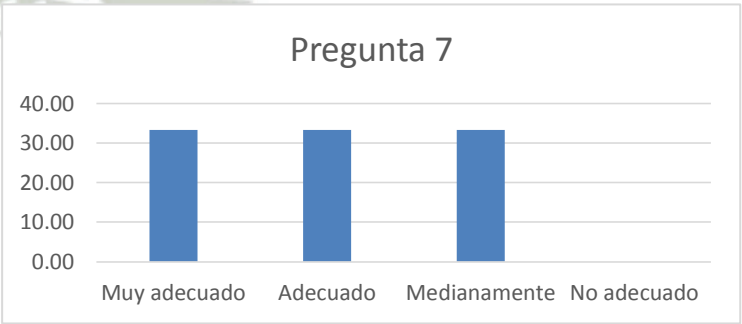
El 22.22% de los entrevistados indicaron que la forma de escalamiento de la técnica era muy buena mientras que el 66.67% indicaron que era buena. Sólo el 11.11% indicó que era mala. En conclusión, el 88.89% manifestaron que la forma de escalonamiento era buena.

PREGUNTA 7: En la técnica propuesta. ¿El ciclo de vida para la implementación de pruebas de software es adecuado?

Tabla 13
Pregunta 7

CARACTERISTICA	CANTIDAD	%
Muy adecuado	3	33.33
Adecuado	3	33.33
Medianamente	3	33.33
No adecuado	0	0.00
TOTAL	**9**	**100.00**

Fuente: Elaborado por el autor



El 33.33% de los entrevistados indicaron que el ciclo de vida de la técnica era muy adecuada mientras que el 33.33% manifestó que era adecuada. El 33.33% de los

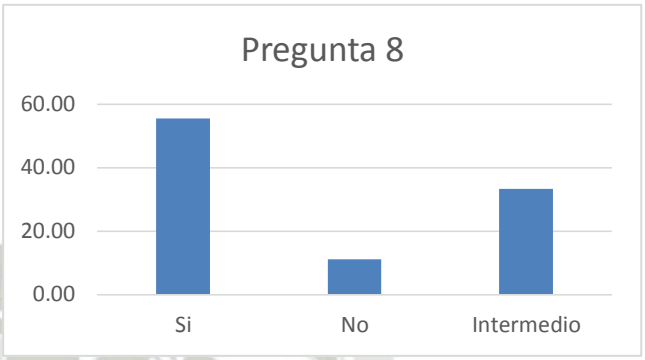
mismos indicaron que era medianamente adecuada. Se concluye que la técnica era adecuada para los entrevistados.

PREGUNTA 8: En la técnica propuesta. ¿La gestión de pruebas de software durante el ciclo de desarrollo de software lleva a cabo la evaluación sugerida por PMBOK?

Tabla 14
Pregunta 8

CARACTERISTICA	CANTIDAD	%
Si	5	55.56
No	1	11.11
Intermedio	3	33.33
TOTAL	**9**	**100.00**

Fuente: Elaborado por el autor



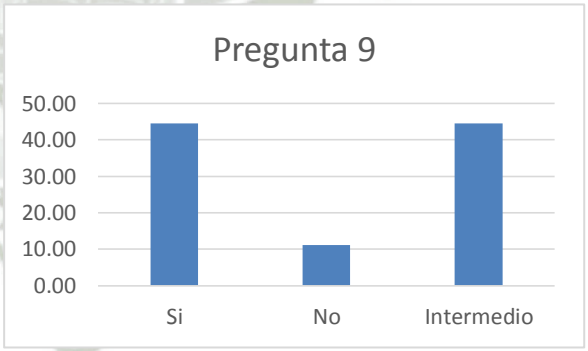
El 55.56% de los entrevistados indicaron que la gestión de pruebas era la sugerida por el PMBOK. El 11.11% indicaron que no lo era mientras que el 33.33% manifestaron un punto intermedio.

PREGUNTA 9: En la técnica propuesta. ¿La gestión de pruebas de software durante el ciclo de desarrollo es la adecuada?

Tabla 15
Pregunta 9

CARACTERISTICA	CANTIDAD	%
Si	4	44.44
No	1	11.11
Intermedio	4	44.44
TOTAL	**9**	**100.00**

Fuente: Elaborado por el autor



El 44.44% de los entrevistados indicaron que la gestión de pruebas de software durante el ciclo de desarrollo si es la adecuada mientras que el 11.11% de los mismos manifestaron que no lo era. El otro 44.44% manifestaron un punto intermedio.

Conclusiones y Recomendaciones

Conclusiones

PRIMERA: La técnica propuesta para implementar pruebas de software queda definida de manera que permita manejar los proyectos desde las primeras etapas del desarrollo de software cuando se trata de software elaborado para Cooperativas de Ahorro y Crédito.

SEGUNDA: La técnica de escalamiento permite que los modelos existentes de pruebas de software, puedan establecer metodologías que sugieren adquirir distintos niveles de madurez en los procesos, sin embargo no especifican particularidades, lo cual conduce a que la implementación resulte compleja.

TERCERO: Las métricas de calidad propuestas para los procesos de pruebas desarrollados al final de la etapa de codificación traen consigo un alto índice de riesgos que se ven reflejados en retrasos en el tiempo, altos costos y baja calidad del producto de software.

CUARTA: Los procesos de pruebas y todos aquellos elementos que apunten a la calidad del producto, permiten obtener elementos para lograr la madurez del modelo de calidad del producto mediante procesos de retroalimentación.

Recomendaciones

PRIMERA: Se recomienda proponer un proceso unificado que profundice y particularice los detalles que los estándares y modelos de calidad no contemplan. El carácter formal de proceso unificado debería basarse en los estándares más importantes y reconocidos en la industria del software.

SEGUNDA: Se recomienda ampliar el estudio para llevar a cabo una relación con otras metodologías o modelos que resuelvan el problema de la calidad en función de los estándares vistos n el mercado.

TERCERA: Implementar una guía de pruebas de calidad propuesta para una gama de diversos tipos de proyectos para demostrar su capacidad de adaptación, ya que esta guía hace énfasis en las buenas prácticas recomendadas para la construcción de software y de la selección más cuidadosa de las pruebas de calidad adecuadas para lograr una mejora continua en estos procesos.

Bibliografía

BARTHELEMY, J. (2001). The hidden cost of IT outsourcing. Sloan Management Review, 42(3), 60-69.

BLACK, A. (2004). Critical Testing Processes: plan, prepare, perform, perfect. Addison Wesley Longman Publishing: New York.

BURNSTEIN, I. (2003). Practical Software Testing. Springer-Verlag: Chicago, USA.

CASTILLO RODRÍGUEZ, OMAR, et al (2000). Administración orientada a Proyectos de Software. Tesis para optar el grado académico de Licenciado en Administración de Sistemas de Información. Universidad Francisco Marroquín. Guatemala.

COHEN, C. F., BIRKIN, S.J., GARFIELD, M. J., and WEBB, H. W. (2004). Managing conflict in software testing. Communications of the ACM, 47(1), 76-81.

CRAIG R.D. and JASKIEL, S.P. (2002). Systematic Software Testing, Artech House Publishers: Boston, EEUU.

DIEZ EDUARDO (2003). Generador del Mapa de Actividades de un Proyecto de Desarrollo de Software. Tesis para optar el grado de magister. Universidad Politécnica de Madrid. Argentina.

DINES BJORNER (2006). Software Engineering 2. Ed. Springer Verlag. 2006.

DONAHOE, N. and PECHT, M. (2003) Are U.S. Jobs Moving to China?. IEEE Transactions on Components and Packaging Technologies. 26(3), 682-686.

EJAZ R. (2006) Software maintenance outsourcing: Issues and strategies. Computers and Electrical Engineering. 32(6), 449-453.

GONZALEZ, R., GASCO, J., LLOPIS, J. (2005). Information systems outsourcing risks: a study of large firms. Industrial Management & Data Systems, 105(1), 45 – 62.

GOSLIN, A. (2009). TMMi Assessment Method Application Requirements (TAMAR). TMMi Foundation: Ireland.

HETZEL, W. (1988). The Complete Guide to Software Testing, QED Information Sciences. Wellesley Inc. Massachussetts.

HUENNAN, MARCO (2003). Software en minería: Aporte en precision, rapidéz y versatilidad. Informe técnico. Enero 2013. Número 379.

KANER, C., FALK J., and NGUYEN, H.Q. (1999) Testing Computer Software. John Wiley & Sons: New York.

KIT, E. (1995). Software Testing in the Real World: Addison-Wesley: Gran Bretaña.

KOIRALA, S., and SHEIKH, S. (2008). Software Testing: Interview Questions. Infinity Science Press: Toronto, Canadá.

KOOMEN, T. (1999) Test process improvement: a practical step-by-step guide to structured testing. AddisonWesley: Great Britain.

LORENZ, MARK AND KIDD, JEFF (1994). Object Oriented Software Metrics, Prentice Hall Publishing, 1994.

MARANTE ESTELLÉS MARÍA (2009). Planificación y seguimiento en proyectos de desarrollo y mantenimiento de software dirigido por la gestión de tiempos. Tesis para optar el grado de magister en Ingeniería de Software. Universidad Politécnica de Valencia. España.

MONTIEL VERA, FRANCISCO (2006). Análisis y estudio comparativo de los programas de computación para la Administración de Proyectos. Tesis para obtener el título profesional de Ingeniero Industrial. Universidad Autónoma del estado de Hidalgo. Instituto de Ciencias Básicas e Ingenierías. México.

McCARTHY I. and ANAGNOSTOU A. (2004). The impact of outsourcing on the transaction costs and boundaries of manufacturing. International Journal of production economics, 88(1), 61-71.

MCCONNELL STEVE (1996). Desarrollo y Gestión de Proyectos Informáticos. 1996.

PRESSMANN ROGER (2005). Ingeniería de Software. Un enfoque Práctico. Mc. Graw Hill. Sexta Edición. 2005.

RANDOLPH ALAN (1993), Barry Posner. Gerencia de Proyectos, Mc Graw Hill. 1993.

SIMON, J.C., POSTON, R.S. and KETTINGER, B. (2009). Creating Better Governance of

Offshore Services. Information System Management. 26(2), 110-122.

SMITE D. (2006) Global Software Development Projects in One of the Biggest Companies in Latvia: Is Geographical Distribution a Problem?. Software Process Improvement and Practice, 11(1), 61-76.

SOMMERVILLE IAN (2007). Ingeniería de Software. Prentice Hall. Addison Wesley. Octava Edición. 2007.

STEINER, M., BLASCHKE, M., PHILIPP, M., and SCHWEIGERT, T. (2010). Make test process assessment similar to software process assessment—the Test SPICE approach. Journal of Software Maintenance and Evolution: Research and Practice, early view, DOI: http://dx.doi.org/10.1002/smr.507.

TAIPALE, O., SMOLANDER, K. and KÄLVIÄINEN, H. (2006). Cost Reduction and Quality Improvement in Software Testing. In the Proceedings of the Software Quality Management Conference, Southampton, United Kingdom.

THAYER RICHARD (1998). Software Engineering Project Management. IEEE Computer Society. 1998.

VARAS MARCELA (1998). Apuntes de clases: Gestión de Proyectos de Ingeniería de Software. Universidad de Concepción, 1998 (en colaboración con Ximena Hormazábal, Luis Monsalve, Jorge Muñoz, César Olivares, Rodrigo Oviedo y Carmen Wolff). 1998.

VEENENDAAL, E. (2010). Test Maturity Model integration (TMMi). TMMi Foundation, Ireland.

WEIGELT, C. (2009). The impact of outsourcing new technologies on integrative capabilities and performance. Strategic Management Journal, 30(6), 595-616.

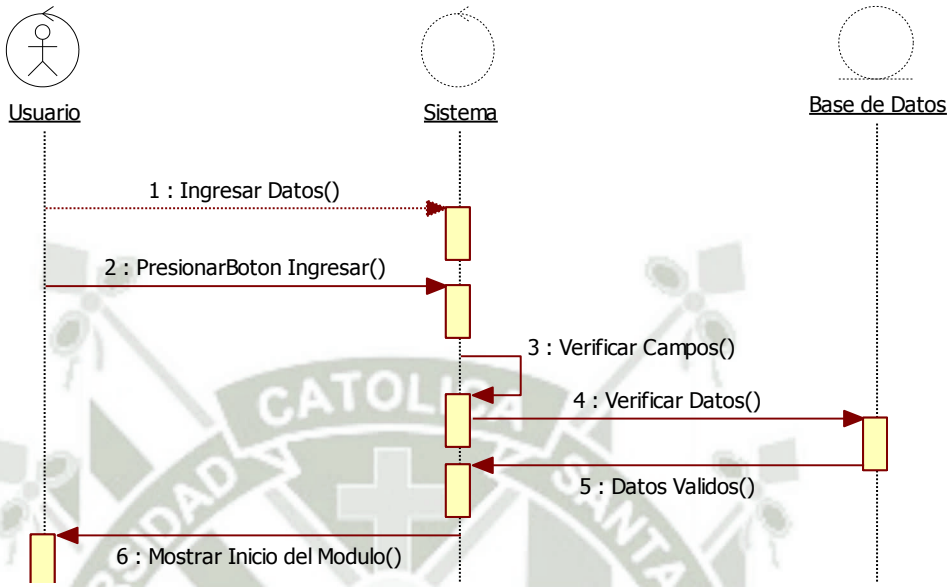
ZHAO J. (2008). Measuring Coupling in Aspect-Oriented Systems. Department of Computer Science and Engineering. Fukuoka Institute of Technology.

ANEXOS

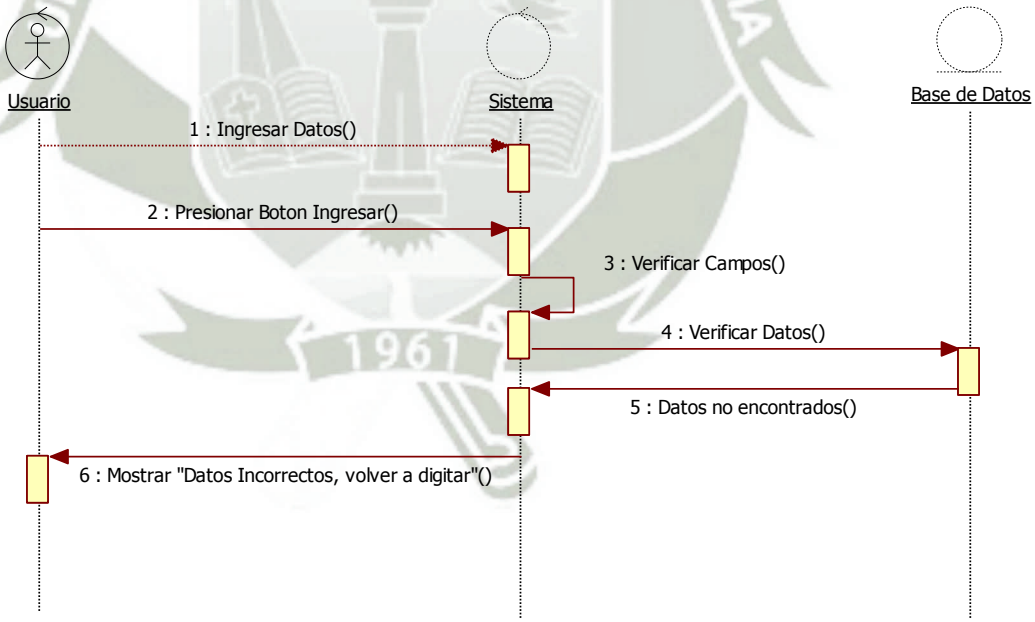
ANEXO N° 1: DIAGRAMA DE SECUENCIAS DEL PRODUCTO

DS001 Login

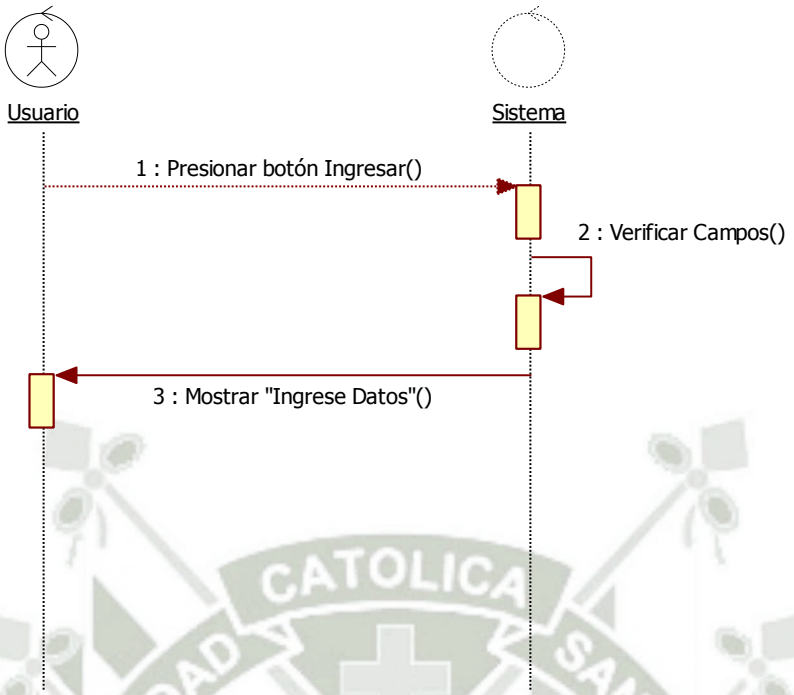
- **Flujo Normal**



- **Flujo Alternativo**

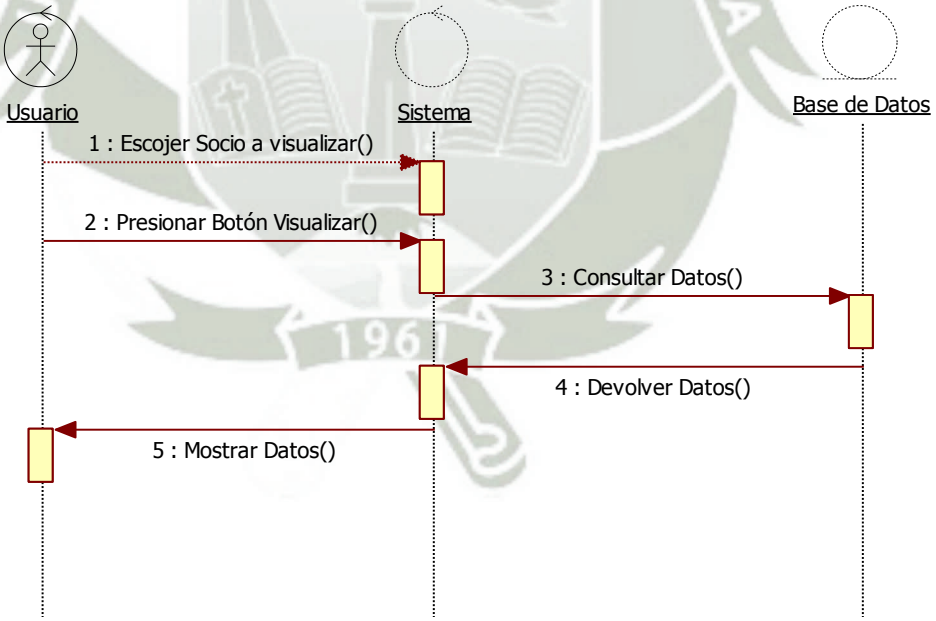


- **Flujo Excepcional**



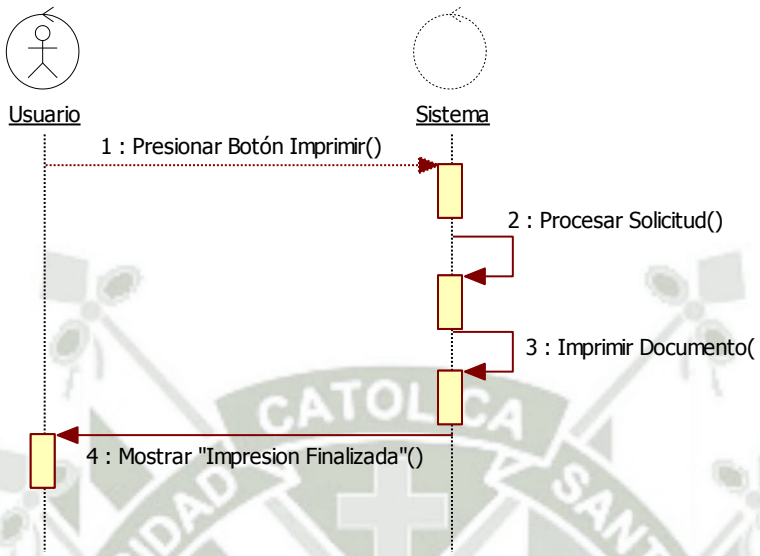
DS002 Consultar Historial Crediticio

- **Flujo Normal**



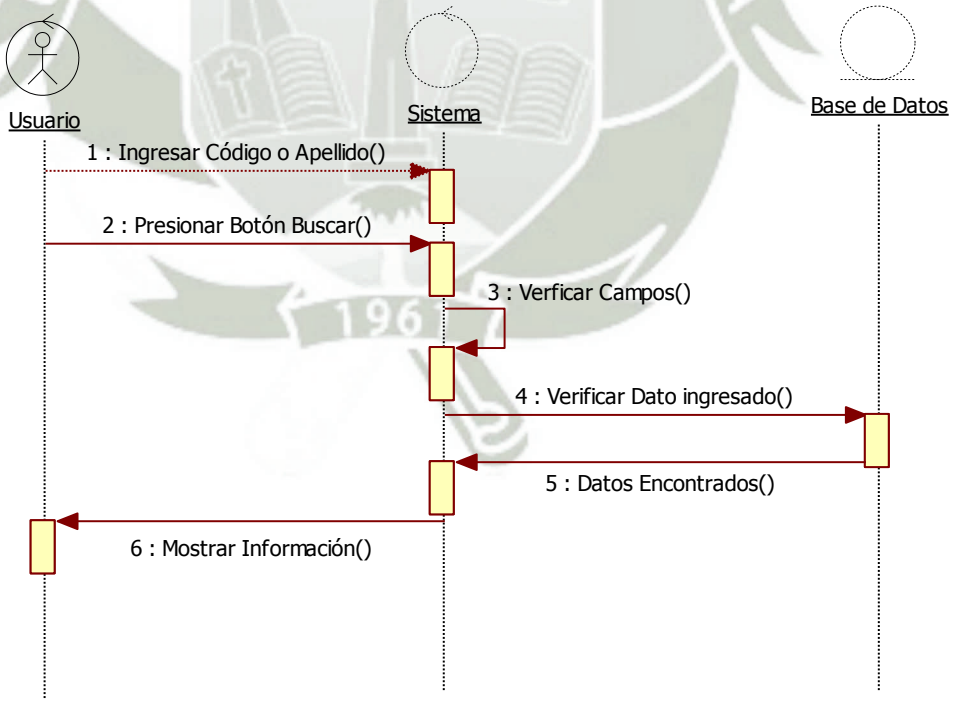
DS003 Imprimir Historial

- **Flujo Normal**

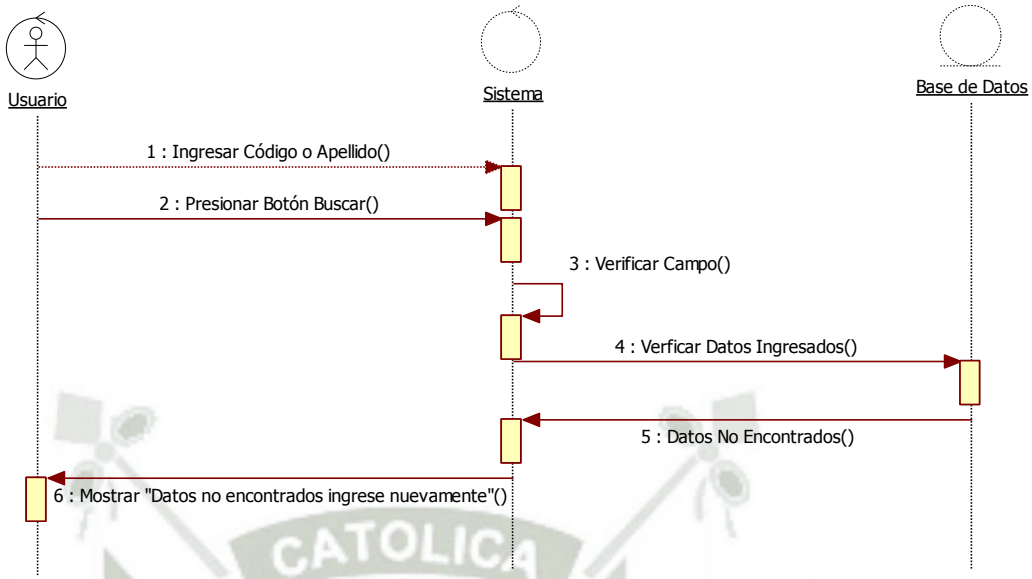


DS004 Buscar Socio

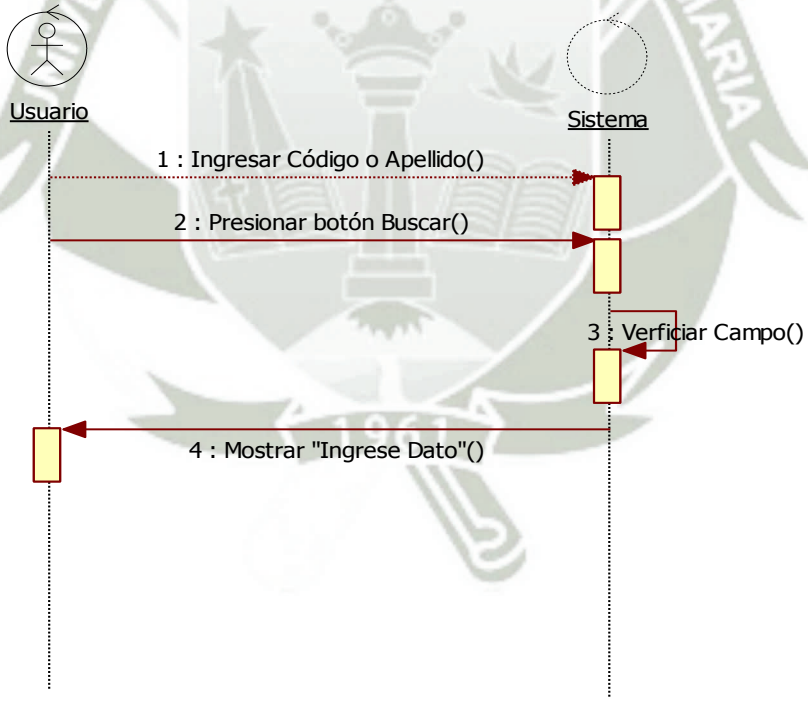
- **Flujo Normal**



- **Flujo Alternativo**

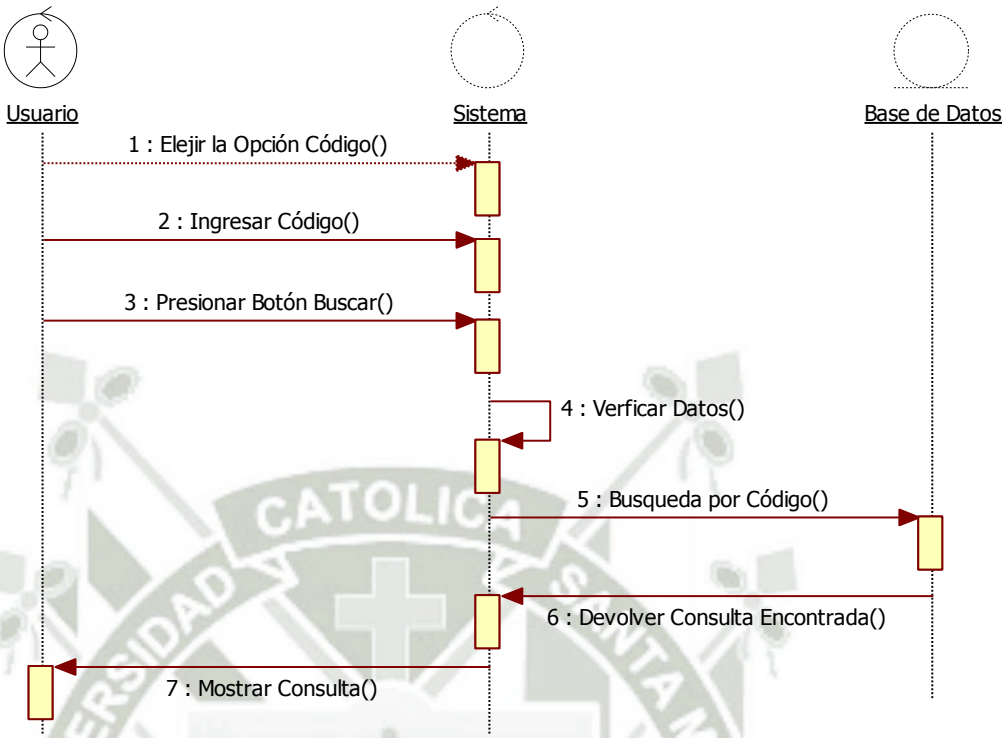


- **Flujo Excepcional**

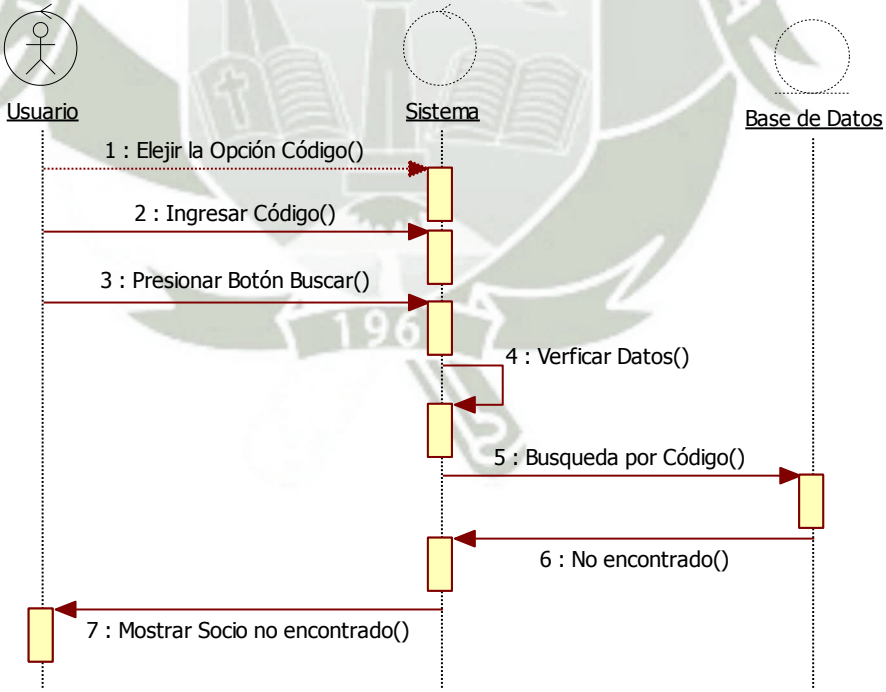


DS005 Buscar por Código

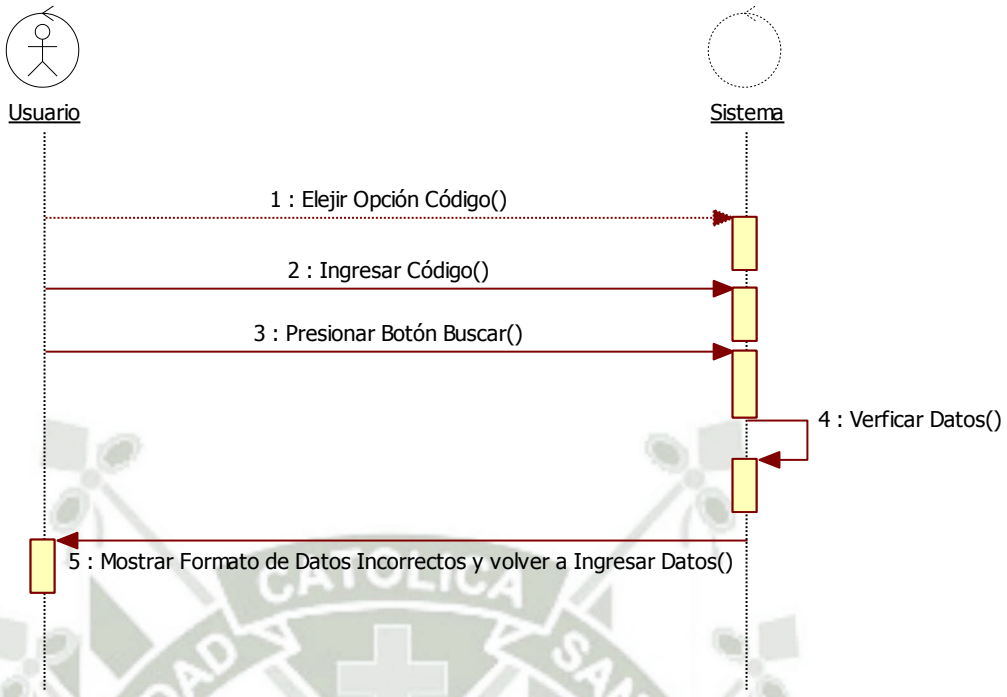
- **Flujo Normal**



- **Flujo Alternativo**

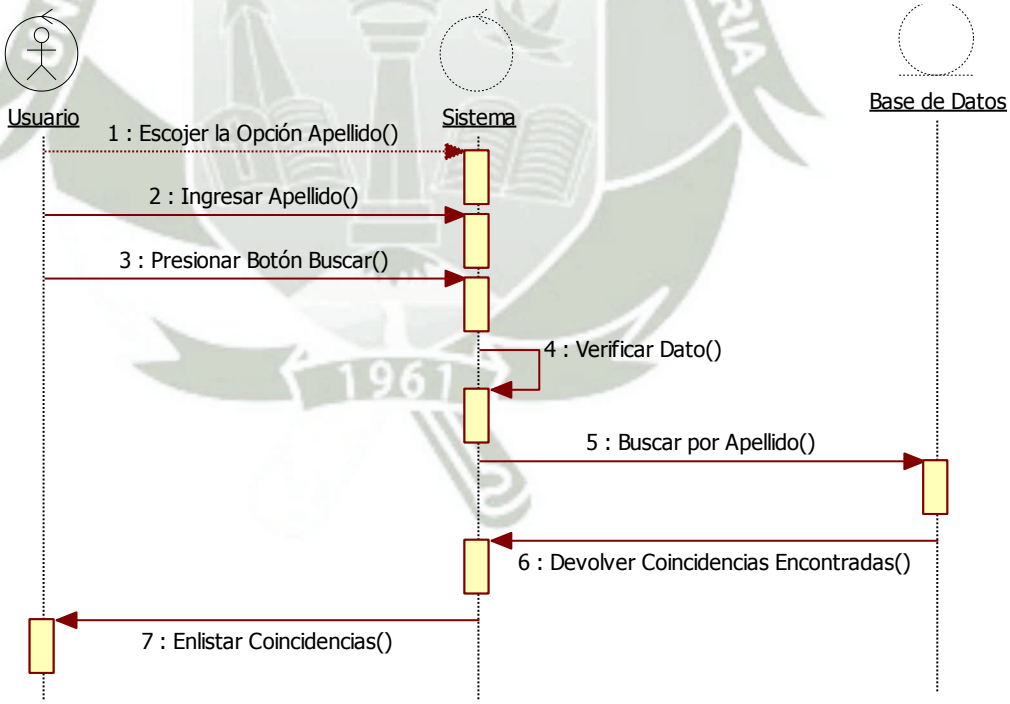


- **Flujo Excepcional**

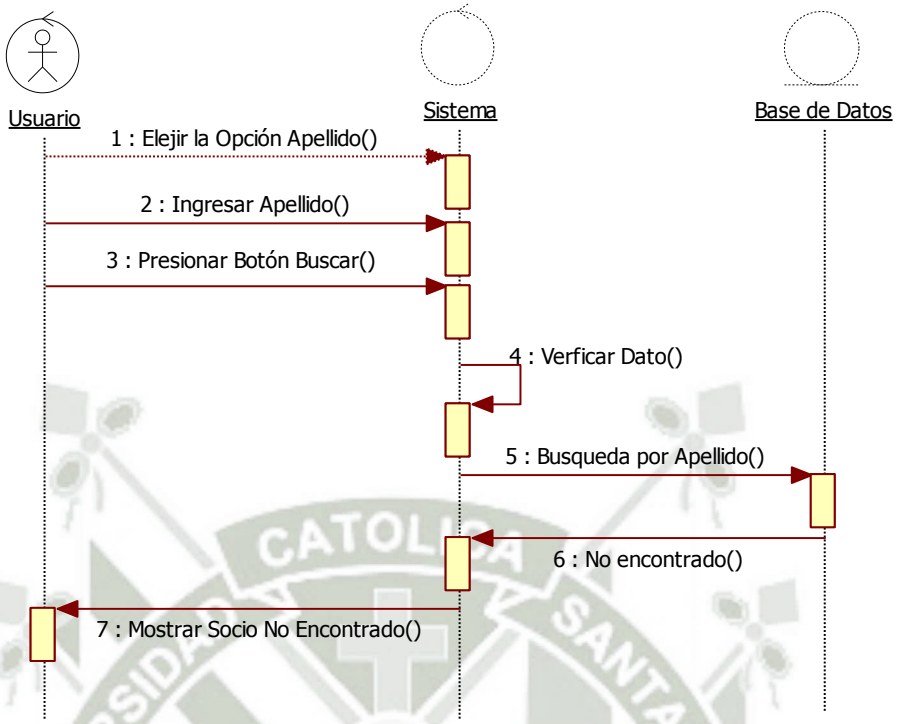


DS006 Buscar por Apellido

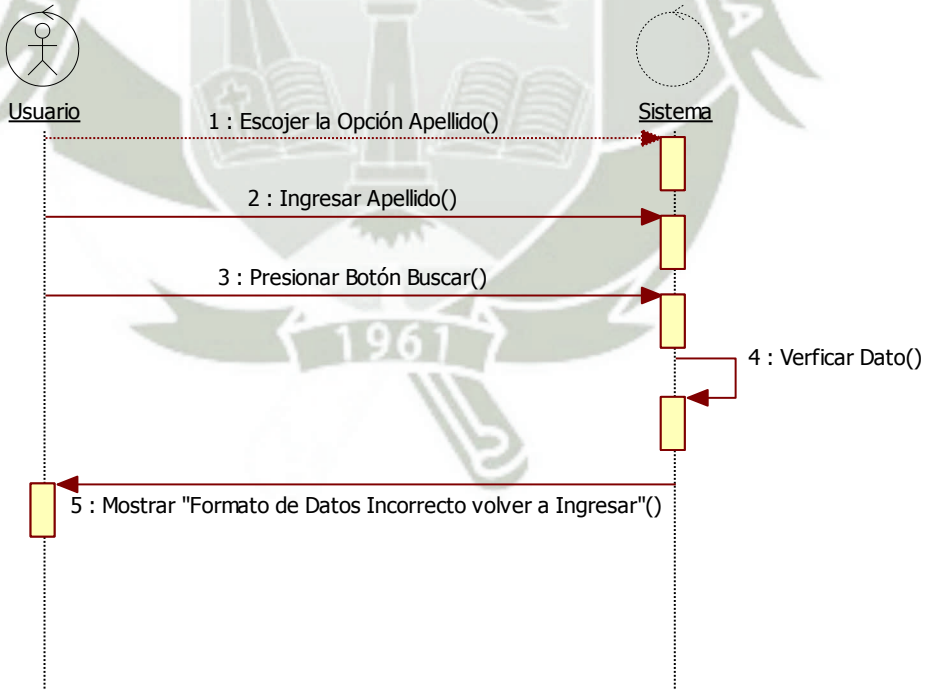
- **Flujo Normal**



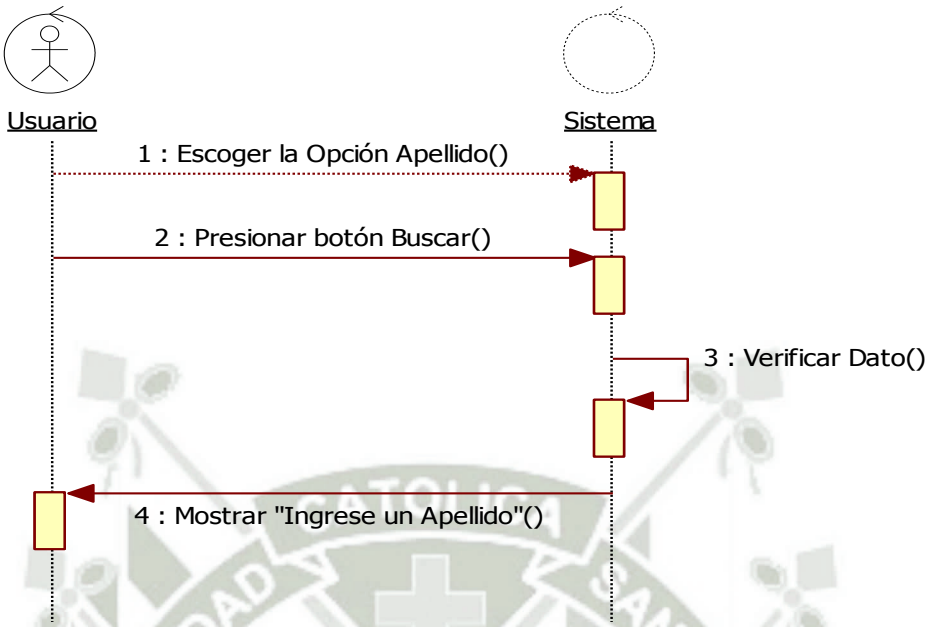
- **Flujo Alternativo**



- **Flujo Excepcional 1**

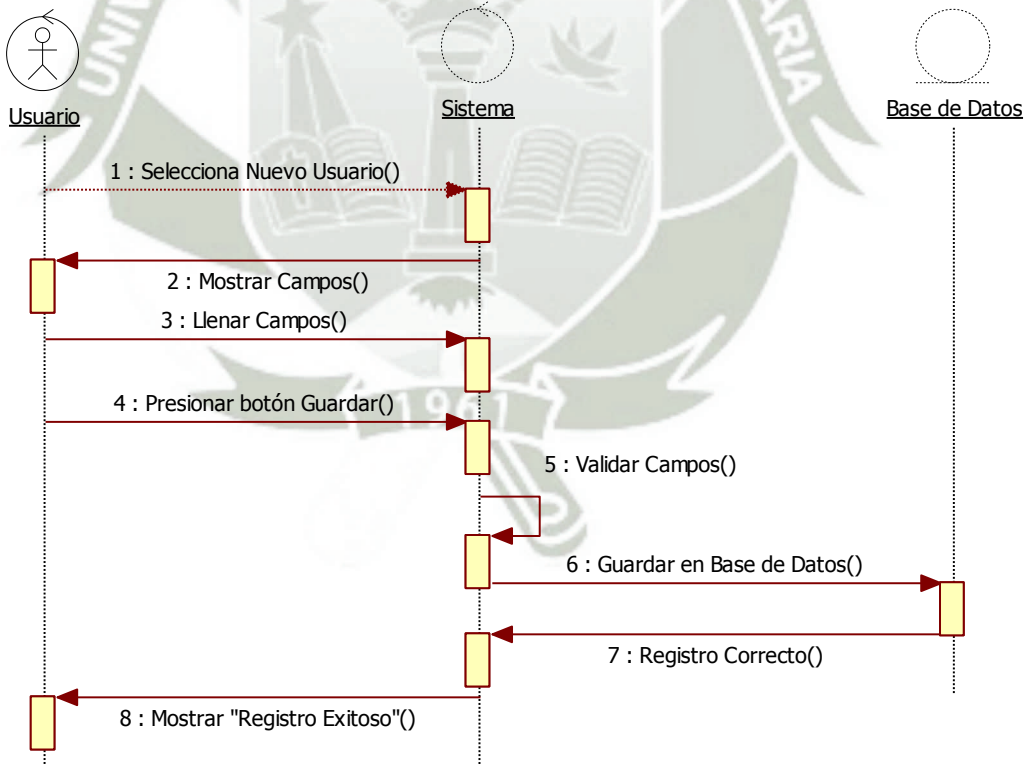


- **Flujo Excepcional 2**

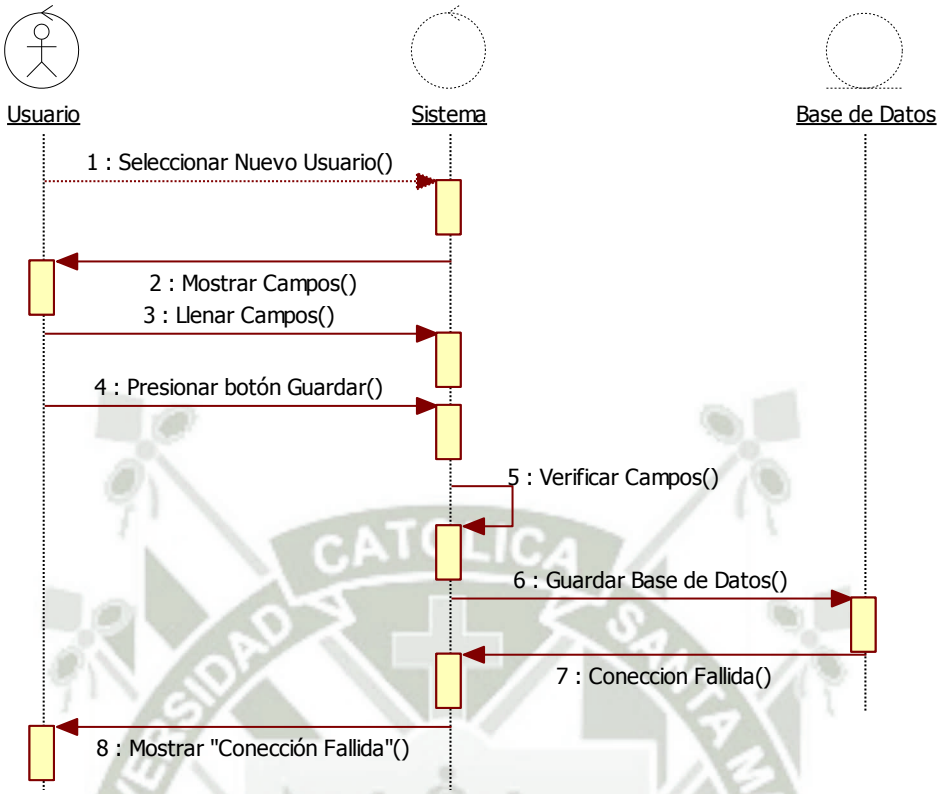


DS007 Registrar Usuario

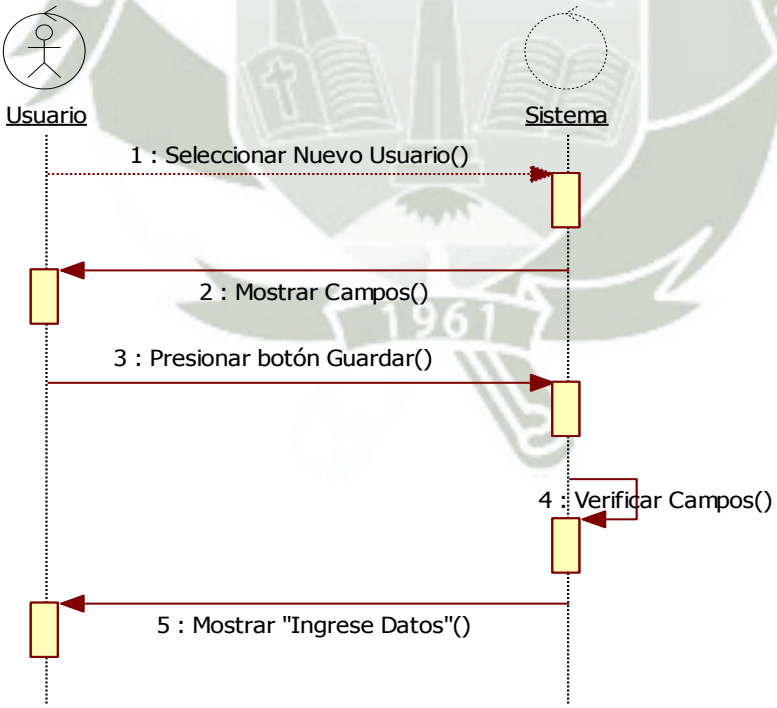
- **Flujo Normal**



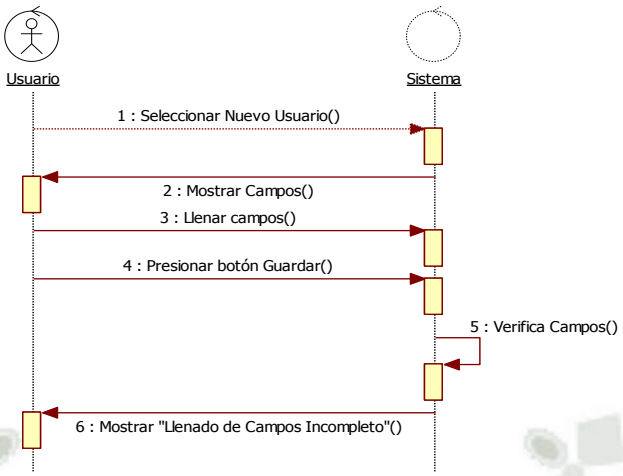
- **Flujo Alternativo**



- **Flujo Excepcional 1**

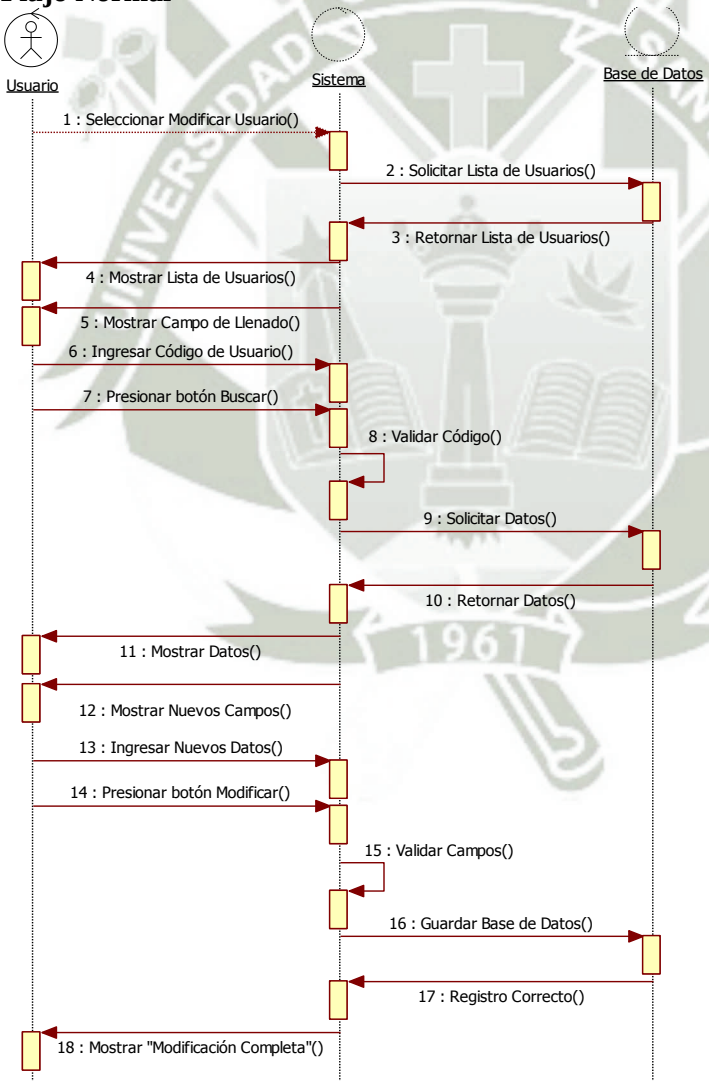


- **Flujo Excepcional 2**

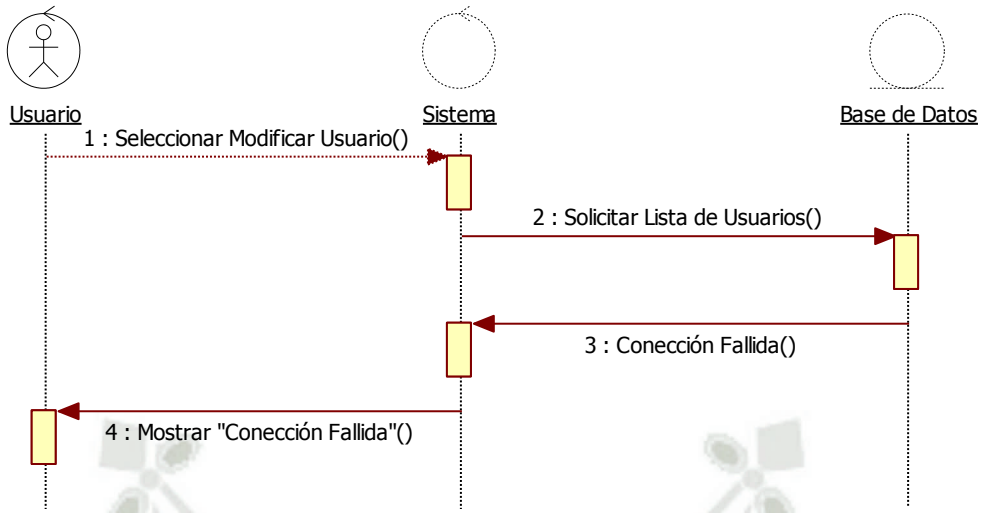


DS008 Modificar Usuario

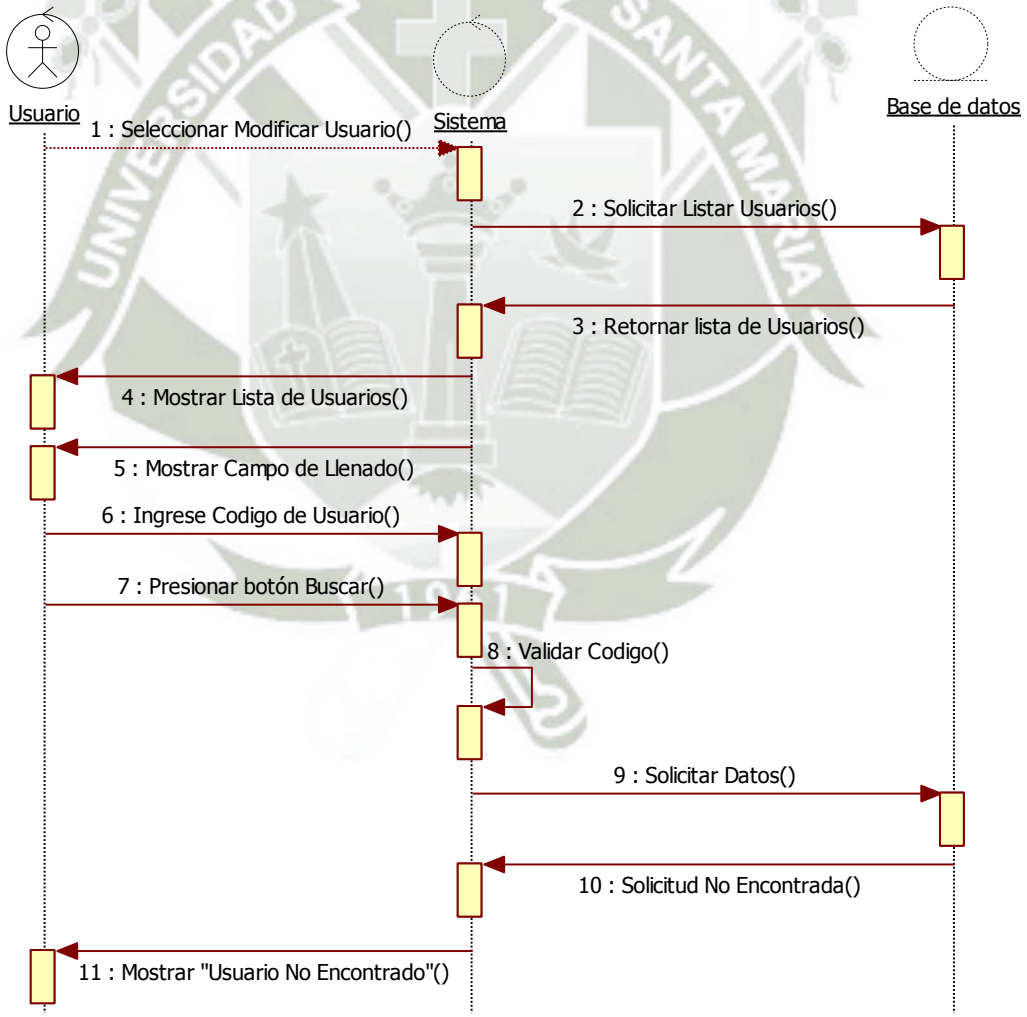
- **Flujo Normal**



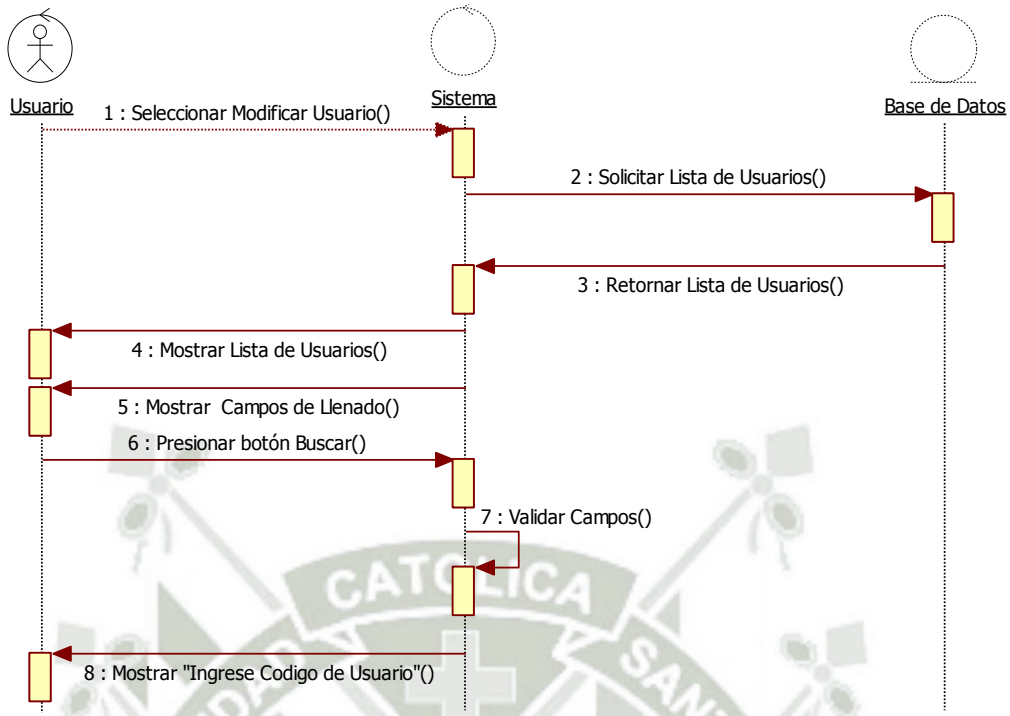
- **Flujo Alternativo 1**



- **Flujo Alternativo 2**

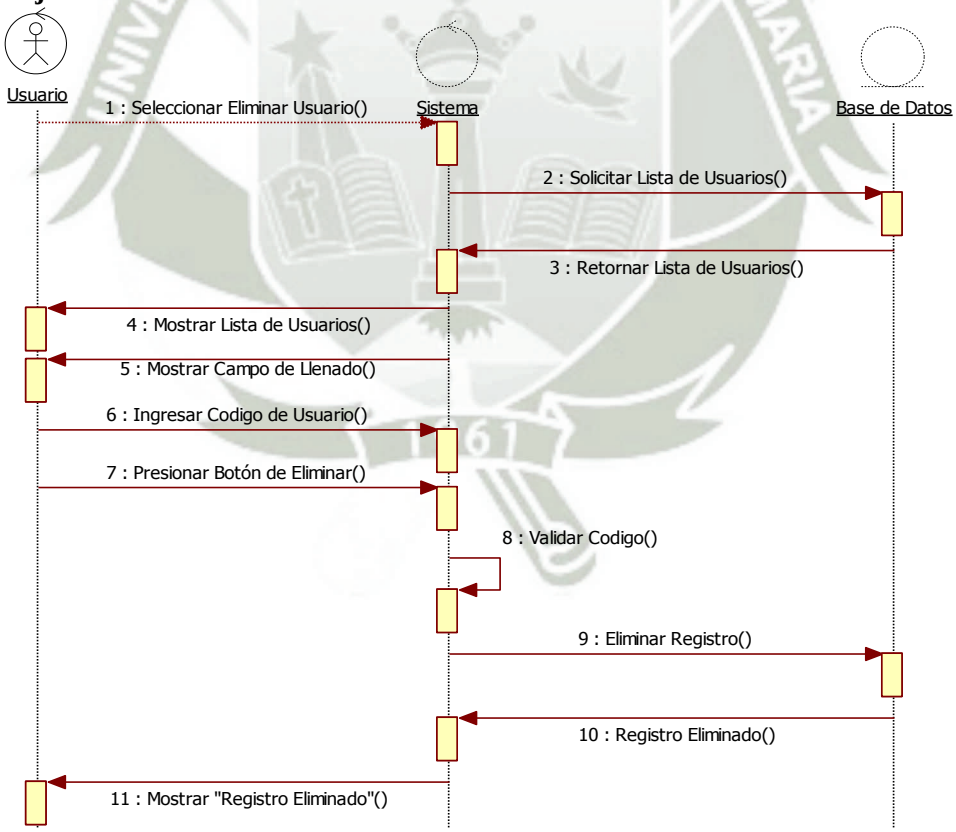


- **Flujo Excepcional**

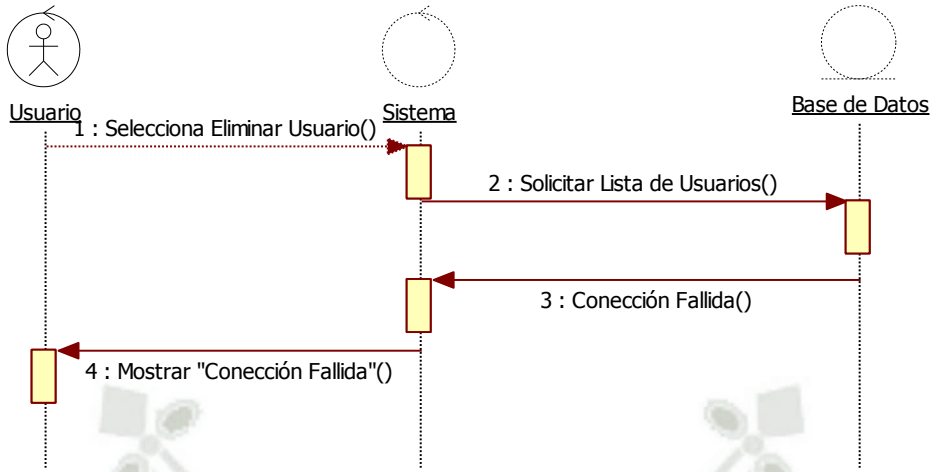


DS009 Elimina Usuario

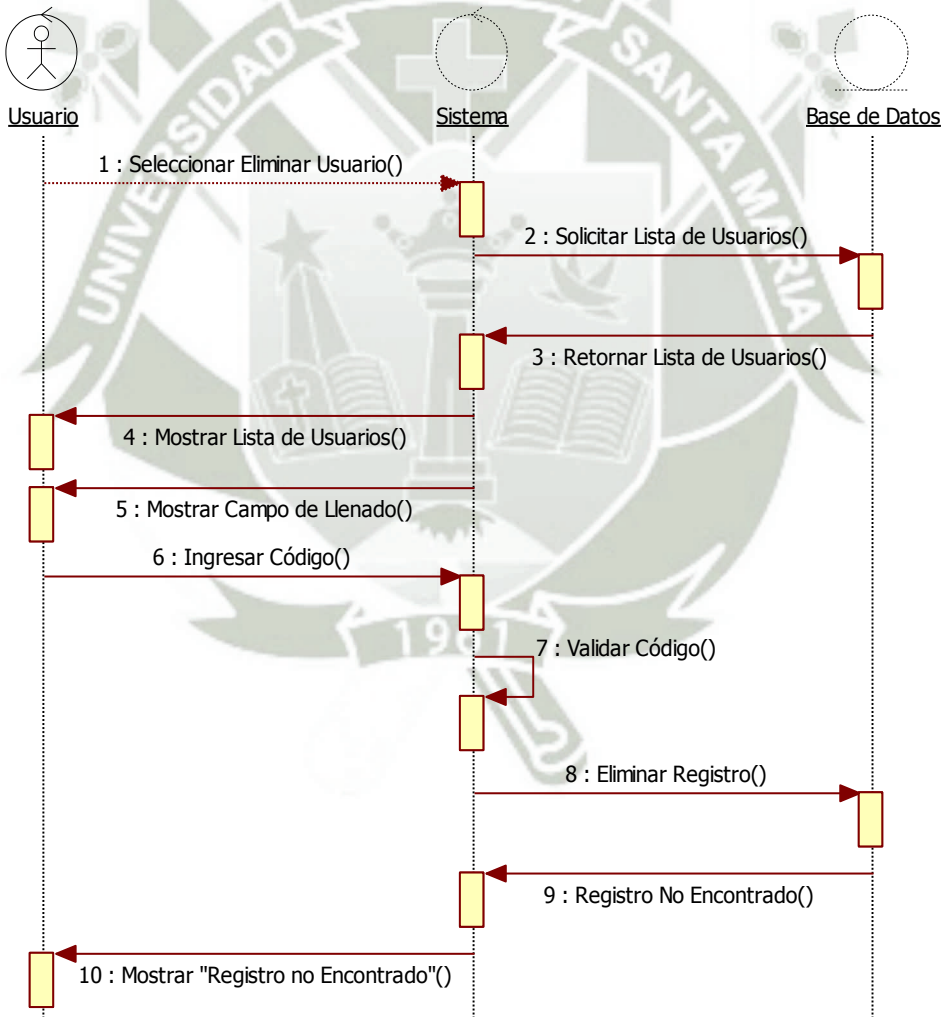
- **Flujo Normal**



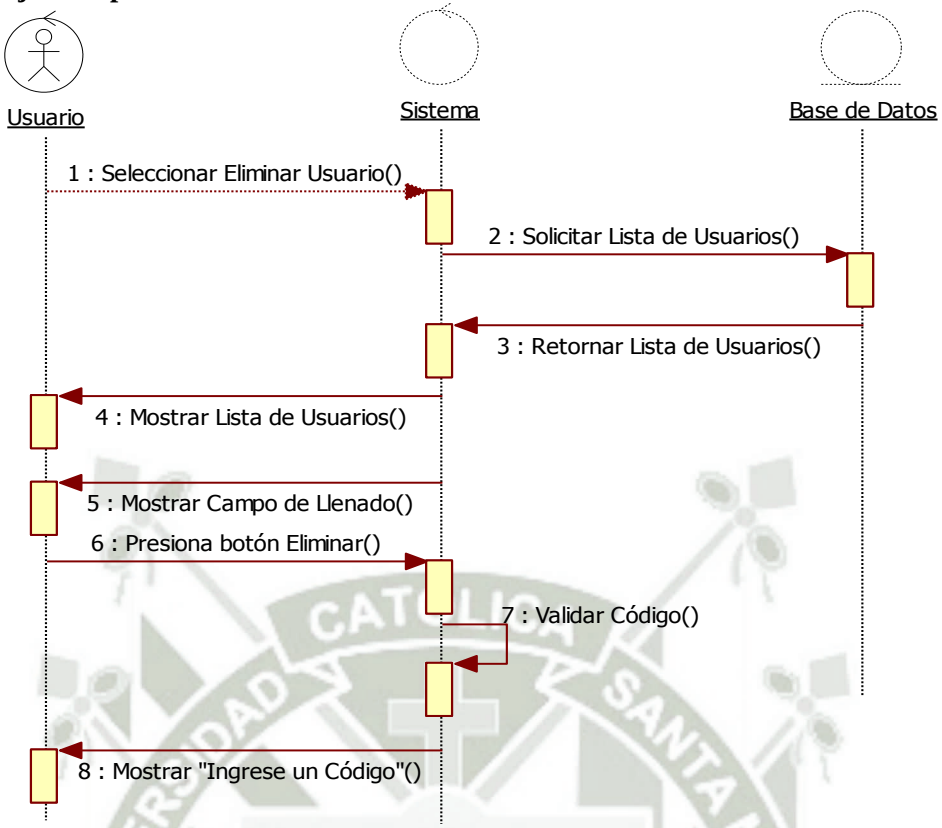
- **Flujo Alternativo 1**



- **Flujo Alternativo 2**

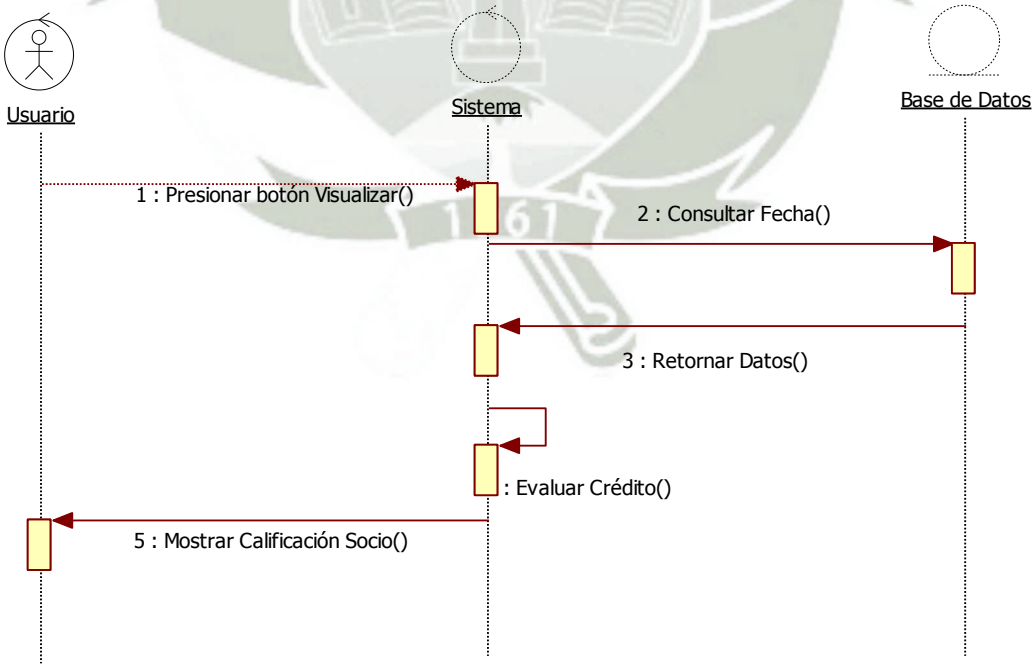


- **Flujo Excepcional**



DS010 Evaluar Historial Crediticio

- **Flujo Normal**



ANEXO N° 2: ENCUESTA PARA VALIDAR LA TECNICA

1. ¿Cómo califica la técnica para detectar las causas a los problemas de las pruebas de software?

   Muy bueno □ Bueno □ Malo □ Pésimo □

2. ¿Cómo califica la definición de las métricas para reconocer el estado de las prácticas en el ciclo de desarrollo de software?

   Muy bueno □ Bueno □ Malo □ Pésimo □

3. ¿Cómo califica al proceso de soporte sugerido para la formulación de pruebas de software?

   Muy fácil □ Fácil □ Difícil □ Muy difícil □

4. ¿Cómo califica usted la técnica propuesta que permite implementar pruebas de software?

   Muy sencilla □ Sencilla □ Complicada □ Muy complicada □

5. ¿La técnica propuesta es fácil de comprender?

   Muy fácil □ Fácil □ Difícil □ Muy difícil □

6. ¿Cómo califica usted la forma de escalamiento de la técnica que permite detectar los problemas en las pruebas de software?

   Muy bueno □ Bueno □ Malo □ Pésimo □

7. En la técnica propuesta. ¿El ciclo de vida para la implementación de pruebas de software es adecuado?

   Muy adecuado □ Adecuado□ Medianamente adecuado□ No adecuado□

8. En la técnica propuesta. ¿La gestión de pruebas de software durante el ciclo de desarrollo de software lleva a cabo la evaluación sugerida por PMBOK?

   Si□ No□ Intermedio□

9. En la técnica propuesta. ¿La gestión de pruebas de software durante el ciclo de desarrollo es la adecuada?

   Si□ No□ Intermedio□