“SNAIL”, UNA METODOLOGÍA HÍBRIDA PARA EL DESARROLLO DE APLICACIONES WEB

Jimmy Rolando Molina Ríos
Mariuxi Paola Zea Ordóñez
Fausto Fabián Redrován Castillo
Nancy Magaly Loja Mora
Milton Rafael Valarezo Pardo
Joofre Antonio Honores Tapia

Ingeniería y Tecnología

3ciencias

SNAIL, UNA METODOLOGÍA HÍBRIDA PARA EL DESARROLLO DE APLICACIONES WEB

Jimmy Rolando Molina Ríos
Mariuxi Paola Zea Ordóñez
Fausto Fabián Redrován Castillo
Nancy Magaly Loja Mora
Milton Rafael Valarezo Pardo
Joofre Antonio Honores Tapia

Editorial Área de Innovación y Desarrollo,S.L.

ÁREA DE INNOVACIÓN Y DESARROLLO, S.L.

C/ Els Alzamora, 17 - 03802 - ALCOY (ALICANTE) info@3ciencias.com

Primera edición: **mayo 2018**

ISBN: **978-84-948690-8-2**

DOI: http://dx.doi.org/10.17993/IngyTec.2018.38

Grupo de Investigación de Ingeniería de Sistemas – G.I.I.S.

Ing. Mariuxi Paola Zea Ordóñez, Mg. mzea@utmachala.edu.ec
Coordinadora del Grupo – G.I.I.S.

Ing. Jimmy Rolando Molina Ríos, Mg. jmolina@utmachala.edu.ec
Coordinador del Proyecto. – P.D.I.S.

Ing. Fausto Fabián Redrován Castillo, Mg. fredrovan@utmachala.edu.ec
Ing. Nancy Magaly Loja Mora, Mg. nmloja@utmachala.edu.ec
Ing. Milton Rafael Valarezo Pardo, Mg. mvalarezo@utmachala.edu.ec
Ing. Joofre Antonio Honores Tapia, Mg. jhonores@utmachala.edu.ec
Miembros del Proyecto – Miembro Profesor Titular

Sr. Luis Fernando Vinces Sánchez lvinces_est@utmachala.edu.ec
Sr. Antonio Steeven Gómez Moreno asgomezm_est@utmachala.edu.ec
Sr. Josías Israel Piña Orozco. jipina_est@utmachala.edu.ec
Sr. Kevin David Correa Elizaldes kdcorrea_est@utmachala.edu.ec
Sr. Fabricio Gustavo García Zerda fggarciaz_est@utmachala.edu.ec
Srta. María José Contento Segarra mjcontento_est@utmachala.edu.ec
Srta. Cinthia Paola Flores Cabrera. cpflores_est@utmachala.edu.ec
Sr. Bryan Santiago Ruiz Abad. bsruiz_est@utmachala.edu.ec
Srta. Mónica Janina Gallegos Chamba mjgallegos_est@utmachala.edu.ec
Sr. Carlos Eduardo Jumbo Parrales cjumbo_est@utmachala.edu.ec
Miembros en Formación

Ing. Fausto Juvenal Loja Mora, Mg.
Investigador Asociado

Proyecto Titulado:

Construcción de un modelo referencial híbrido para la gestión de procesos de desarrollo ágil de software web.

Aprobado: H. Consejo Universitario N.º 396/2016.

Fausto, Jimmy, Nancy, Mariuxi, Milton, Joofre

ÍNDICE

CAPÍTULO 1: RESUMEN E INTRODUCCIÓN 13
1.1. Resumen 13
1.2. Introducción 13
CAPÍTULO 2: FASE DE LA METODOLOGÍA SNAIL 15
2.1. Estructura del modelo SNAIL 15
2.1.1. ¿Qué es la metodología SNAIL? 15
2.1.2. Objetivos de SNAIL 15
2.1.3. Características 15
2.1.4. Modelo SNAIL 16
2.1.5. Proceso de SNAIL 16
2.1.5.1. Fase I: Requisitos 18
2.1.5.2. Fase II: Planificación 19
2.1.5.3. Fase III: Diseño 19
2.1.5.4. Fase IV: Programación 19
2.1.5.5. Fase V: Pruebas 20
2.1.5.6. Fase VI: Clausura 20
2.1.5.7. Fase VII: Inbound Marketing 20
CAPÍTULO 3: FASE DE REQUISITOS 21
3.1. Fase de requisitos 21
3.1.1. Modelado de negocio 23
3.1.1.1. Identificación de los procesos de negocio y los procesos de mantenimiento 23
3.1.1.2. Identificar los usuarios, departamentos o elementos de la organización implicados en el proceso de negocio 24
3.1.1.3. Descripción textual del proceso de negocio 24
3.1.1.4. Construir un diagrama de actividades que represente el proceso de negocio 24
3.1.1.5. Especificar las actividades que aparecen en el diagrama de actividades 25
3.1.1.6. Especificar las informaciones que fluyen en el diagrama de actividades 25
3.1.1.7. Reglas del negocio 26
3.1.2. Estudio de factibilidad 29
3.1.2.1. Esquema explicativo para el estudio de factibilidad 29
3.1.2.1.1. Objetivos 29
3.1.2.1.2. ¿Para qué sirve el Estudio de Factibilidad? 30
3.1.2.1.3. ¿Quiénes participan en la etapa? 30
3.1.2.1.4. Actividades de la etapa 30
3.1.2.1.5. El estudio de factibilidad contiene 30
3.1.2.1.6. Descripción de los principales componentes del Estudio de Factibilidad 30
3.1.2.1.6.1. Reconocimiento general del sistema 30
3.1.2.1.6.2. Recursos requeridos 31
3.1.2.1.6.3. Usuarios del sistema 31
3.1.2.1.6.4. Beneficios esperados del proyecto 31
3.1.2.1.6.5. Costos del proyecto 32

3.1.2.1.6.6. Análisis de alternativas de implementación 33
3.1.2.1.6.7. Análisis de factibilidad del sistema 33
1.3.2.1.6.8. Definir el impacto sobre los planes generales de la organización o sobre su planeación estratégica 34
1.3.2.1.6.9. Cronograma 34
3.1.3. Identificar actores o usuarios y requisitos 36
3.1.3.1. FASE 1: Recolección 37
3.1.3.1.1. Entrevistas: 38
3.1.3.1.2. Casos de Uso y/o Escenarios 39
3.1.3.2. FASE 2: Análisis 39
3.1.3.2.1. JAD (Joint Application Development) 40
3.1.3.2.2. Priorización de Requerimientos 41
3.1.3.2.2.1. Modelos 41
3.1.3.2.2.2. Características de un Buen Requerimiento 42
3.1.3.2.2.2.1. Características de la Descripción de un Requerimiento 42
3.1.3.2.2.2.2. Características de la Especificación de Requerimientos 43
3.1.3.3. Fase 3: Documentación 43
3.1.3.4. Fase 4: Verificación 44
3.1.3.5. Herramientas, técnicas y software 44
3.1.4. Identificar requisitos funcionales y no funcionales 46
3.1.4.1. Recolección de requerimientos 46
3.1.4.1.1. Recolección de objetivos del sistema 47
3.1.4.1.2. Descomposición de los objetivos del sistema en requerimientos de usuario 48
3.1.4.1.2.1. Casos de uso 48
3.1.4.1.2.1.1. Plantillas 48
3.1.4.1.2.1.2. Escenarios 52
3.1.4.1.3. Reorganización de requerimientos 52
3.1.4.1.4. Refinamiento De Requerimientos 52
3.1.5. Analizar los requerimientos 52
3.1.6. Validar requisitos 54
3.1.6.1. Revisión de pares 55
CAPÍTULO 4: FASE DE PLANIFICACIÓN 59
4.1. Fase de planificación 59
4.1.1. Selección de requisitos o historias de usuario 59
4.1.2. Definir entregables 59
4.1.3. Estimación de costos 60
4.1.3.1. Modelo basado en SLOC 60
4.1.3.2. Modelo no basado en SLOC 61
4.1.3.3. Modelo basado en puntos de casos de uso 65
4.1.4. Velocidad del proyecto 70
4.1.5. Escenarios de Bechmarking 71
4.1.5.1. ¿Cómo se realiza? 72
CAPÍTULO 5: FASE DE DISEÑO 77
5.1. Fase de diseño 77
5.1.1. Diseño de bases de datos 79
5.1.2. Diccionario de datos 80

5.1.2.1. Contenido de un Diccionario de Datos 81
5.1.2.2. Notación del Diccionario de datos 81
5.1.3. Diseño conceptual 83
5.1.3.1. Implementación de capa conceptual 84
5.1.4. Diseño Navegacional 84
5.1.4.1. Implementación de capa Navegacional 85
5.1.5. Diseño de interfaz abstracta 85
5.1.5.1. Implementación de capa abstracta 85
CAPÍTULO 6: FASE DE PROGRAMACIÓN 91
6.1. Fase de programación 91
6.1.1. Codificación 91
6.1.1.1. Recodificación 92
6.1.1.2. Programación en pares 93
6.1.1.3. Disponibilidad del cliente 93
6.1.2. Estándares 94
6.1.3. Prueba Unitaria 94
6.1.3.1. Programación dirigida por las pruebas 94
6.1.3.2. Detección y corrección de errores 95
6.1.4. Interconexión 95
6.1.5. Integración 95
6.1.5.1. Integración permanentes 95
CAPÍTULO 7: FASE DE PRUEBAS... 99
7.1. Fase de pruebas 99
7.1.1. Pruebas Unitarias o de Componente 99
7.1.2. Pruebas de Integración 99
7.1.3. Pruebas de Sistema 99
7.1.4. Detección y corrección de errores 100
7.1.5. Metodología de pruebas 100
7.2. Pruebas de aceptación 101
7.3. Análisis de benchmarking 102
7.3.1. Verificar y Ajustar Regularmente la Estrategia de Implementación en el proyecto 102
7.3.2. Identificar Nuevas Oportunidades para Nuevos Procesos de Benchmarking 102
CAPÍTULO 8: FASE DE CLAUSURA 107
8.1. Fase de clausura 107
8.1.1. Presentar entregables 109
8.2. Evaluación iteral del proyecto 111
8.2.1. Pasos para realizar la evaluación iteral del proyecto 111
8.2.1.1. Determinar el ámbito de iteración 111
REFERENCIAS BIBLIOGRÁFICAS 117

ÍNDICE DE TABLAS

Tabla 1: Fases y Actividades de la metodología SNAIL 17
Tabla 2: Plantilla para especificar el diagrama de actividad 1 25
Tabla 3: Plantilla para especificar el diagrama de actividad 2 26
Tabla 4: Herramientas que se utilizan para las diferentes fases de la ingeniería de requerimientos 44
Tabla 5: Características de las herramientas de Administración de requerimientos más utilizadas 45
Tabla 6: Plantilla para la descripción de requerimientos funcionales 48
Tabla 7: Plantilla para la descripción de requerimientos no funcionales 51
Tabla 8: Nivel de complejidad para los ILF y EIF 62
Tabla 9: Nivel de complejidad para EI, EO y EQ 63
Tabla 10: Cuenta total de puntos de función 63
Tabla 11: Factores de complejidad 64
Tabla 12: Clasificación y peso de los actores 66
Tabla 13: Clasificación y peso de los casos de uso 66
Tabla 14: Factores técnicos 67
Tabla 15: Factores ambientales 68
Tabla 16: Ejemplo 1 de Historias de Usuario 73
Tabla 17: Ejemplo 2 de Historias de Usuario 73
Tabla 18: Ejemplo de benchmarking en “usabilidad” 75
Tabla 19: Pruebas de aceptación 103
Tabla 20: Caso de prueba de acceso al sistema 103
Tabla 21: Caso de prueba creación de permiso 104
Tabla 22: Caso de prueba gestión de usuario 104
Tabla 23: Caso de prueba registrar docente 105
Tabla 24: Entregable del proyecto 115
Tabla 25: Entregables de control de calidad 116

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Evolución de los largos ciclos de desarrollo (a) a ciclos iterativos más cortos (b) y a la mezcla que hace SNAIL 16
Ilustración 3: Proceso de la Ingeniería de Requerimientos 37
Ilustración 2: Fases del modelo SNAIL 18
Ilustración 4: Modelo de proceso para Recolección de Requerimientos 46
Ilustración 5: Modelo de proceso para análisis de requerimientos 53
Ilustración 6: Proceso de verificar y validar requerimientos 56
Ilustración 7: Proceso de desarrollo de la base de datos 79
Ilustración 8: Paquete de interfaz con la base de datos, dentro del Diseño Conceptual 86
Ilustración 9: Instanciación de una subclase concreta de EntidadAbstracta 87
Ilustración 10: Construcción de un nodo de la capa navegacional 87
Ilustración 11: Generación de un documento HTML a partir de una fuente XML + XSL 88
Ilustración 12: Proceso de la programación en parejas 97

PARTE I
INTRODUCCIÓN

CAPÍTULO 1: RESUMEN E INTRODUCCIÓN

1.1. Resumen

En este libro se describe una guía para aquellas personas interesadas en aplicar una metodología híbrida para el desarrollo de Sitios de web, la cual va de la mano con la ingeniería de software para el desarrollo de las buenas prácticas. La cual surge como resultado del análisis y fusión de las metodologías actuales y que será una base estable para implementación en el desarrollo de un Sitio web. El proceso de desarrollo de software es una tarea que requiere esfuerzo, dedicación, organización y disciplina. Hoy en día, los especialistas desarrolladores de software cuentan con un sinnúmero de opciones con respecto a metodologías, y su elección será de acuerdo a variables como recursos disponibles, tipo, magnitud de proyecto y principalmente el si se ajusta a los objetivos generales de desarrollo. Una de las áreas con mayor crecimiento en los últimos años ha sido la del desarrollo de software web, debido a que los beneficios reales tanto para empresas como para los clientes se producen cuando las primeras son capaces de integrar por completo su sistema, con una buena estructura y cumpliendo de manera eficiente.

1.2. Introducción

Desde que su inicio, aparición, o aplicación, constantemente se escucha comentar sobre las metodologías, pero poco se conocen de manera específica o del todo. ¿Qué son?, ¿para qué funcionan?, ¿cómo sirven?, ¿De qué manera evolucionan?, ¿De qué manera se desarrollan?, ¿cuál es su fundamento?, son varias de las comunes interrogantes que muchas personas que quieren adentrarse a este mundo, se cuestionan, pero que pocas veces nos logran tener en claro.

Antes de empezar a leer el contenido del presente libro, se debe tener en claro el término metodología, debido a que es de suma importancia que se conozca a la perfección para el entendimiento de lo que se va está desarrollando y lo que usted va a aplicar. Una metodología, según [1] “es la manera en la que se analiza sistemática y teóricamente los métodos que se deseen aplicar al campo de estudio. Busca entender el análisis teórico del grupo de métodos, así como sus principios asociados, en una rama específica del conocimiento como tal. De manera general, la metodología abarca conceptos diferentes pero que son necesarios como, el modelo teórico, el paradigma, las fases y técnicas cuantitativas y cualitativas” en otras palabras una metodología expone dentro de los principios que se manejan en las prácticas de la investigación. Explica el por qué se usan ciertos métodos o herramientas de forma determinada, en un estudio específico.

Las metodologías fueron diseñadas con integridad, aprovechando el conocimiento empírico y teórico que se ha obtenido a lo largo de un período objetivo de tiempo, además, sus recomendaciones se basan en determinadas tradiciones profesionales

que están sujetas a los parámetros de revisión vigentes que se analizaron en el marco de determinada situación operativa, y los cuales se sometieron a un profundo proceso de validación científica; entonces ya explicado estas recomendaciones, los resultados no sólo serán más eficientes, eficaces y efectivos, si no que al mismo tiempo serán más objetivos y la posibilidad de error será mínima.

No hay que olvidar que las metodologías se componen generalmente de tal manera que garanticen la ejecución de una secuencia de "pasos" parcialmente concatenados, los cuales hacen posible la revisión del propio proceso de investigación en sí, pero también de cada uno de los resultados en cada una de las etapas.

Para que todo esto sea realizado a la perfección, las metodologías se someten sistemáticamente a un proceso de revisión y actualización, las cuales deben responder ante los períodos de evolución de cada una de las ramas a las que su objetivo planteado vaya dirigido, por ende, una metodología no sólo está dirigida hacia los conceptos científicos llevándolo a la altura del estado del arte, si no que por sobre todas las cosas, persigue ganar en objetividad de las conclusiones científicas.

En la actualidad existen variadas cantidades de metodologías orientadas al desarrollo de software, de las cuales se pueden agrupar en dos importantes categorías: las metodologías tradicionales y las ágiles. Las metodologías tradicionales aprueban las buenas prácticas que poseen los procesos dentro de la Ingeniería de Software; pero cabe resaltar que para esto es necesario de manejarse con mucha disciplina para continuar con el riguroso proceso que éstas deben de tener. [2]

En cambio, las metodologías ágiles muestran una respuesta de manera rápida al cambio y son más elásticas, aunque cae destacar que generan muy poca documentación y no poseen documentos formales o no los usan.

En el presente documento se propone y elabora una metodología para desarrollo de proyectos de software web en base a la nueva tendencia en el área de Ingeniería de Software: las metodologías híbridas.

Estas metodologías híbridas poseen algunas prácticas existentes tanto en las metodologías tradicionales como en las ágiles, aportando así una gran ventaja. Para el desarrollo y creación de la metodología propuesta, el primer paso consistió en investigar la factibilidad de utilizar una metodología híbrida en el contexto actual de las empresas desarrolladoras de software. Una vez que se comprobó por medio de un las respectivas investigaciones bibliográficas y la tendencia que poseen, se diseñó la metodología, la cual considera las necesidades de dichas investigaciones y combina algunas prácticas existentes dentro de las metodologías RUP (Rational Unified Process, Proceso Unificado de Rational) [3], XP (eXtreme Programming, Programación Extrema) [4], Scrum [5] y OOHDM (Object Oriented Hypermedia Design Model) [6]

CAPÍTULO 2: FASE DE LA METODOLOGÍA SNAIL

2.1. Estructura del modelo SNAIL

El desarrollo de esta metodología nace de la necesidad de abarcar las fases más importantes de otros modelos de realización de software que son cruciales para el buen desarrollo de sistemas formando así un modelo hibrido capaz de combinar las actividades de otras metodologías para que de esta manera las empresas o desarrolladores sean capaces de producir un software orientado a la web, de manera eficiente.

2.1.1. ¿Qué es la metodología SNAIL?

SNAIL (Software Nativo de Arquitectura Iterativa Lógica), es una metodología híbrida de desarrollo de aplicaciones web, que se basa en la simplicidad, comunicación y planificación del código desarrollado, su nombre nace de la forma que tiene el modelo de sus fases, ya que, al ser un modelo en espiral, toma una forma similar a la de un caracol.

2.1.2. Objetivos de SNAIL

- La satisfacción del cliente
- Potenciar el trabajo
- Disminuir el riesgo de manera efectiva sobre las variables del proyecto: costo, tiempo, calidad, alcance.

2.1.3. Características

- Metodología híbrida basada en pruebas y error para la creación de sistemas orientados a la web, funcionales.
- Fundamentada en principios
- Reduce el coste del cambio en todas las etapas del ciclo de vida del sistema.
- Mezcla las mejores prácticas que se han obtenido para desarrollar software y mediante su ciclo en manera de esfera procura llevarlas al su mejor desarrollo.
- Cliente bien definido
- Los requisitos pueden cambiar.

2.1.4. Modelo SNAIL

La metodología SNAIL define cuatro variables para cualquier proyecto de software orientado a la web: costo, tiempo, calidad y alcance, Además de estas cuatro variables, solo 3 de ellas serán fijadas de manera arbitraria por los actores externos al grupo de desarrolladores (clientes y jefes de proyecto). El valor de la variable restante podrá ser definido por el equipo de desarrollo, en función de los valores de las otras tres.

Ej. ***"Si el cliente define el alcance y la calidad, y el jefe de proyecto el precio, el grupo de desarrollo podrá tener toda la libertad para definir el tiempo que el proyecto durará."***

Por esto se realiza ciclos de desarrollo cortos (Iteraciones), con entregables funcionales al finalizar cada ciclo. En cada iteración se realiza un ciclo completo de Requisitos, Panificación, Diseño, Programación, Pruebas y Clausura, siendo un plus opcional darle una fase de Inbound marketing, para su difusión y publicidad, pero utilizando un conjunto de reglas prácticas.

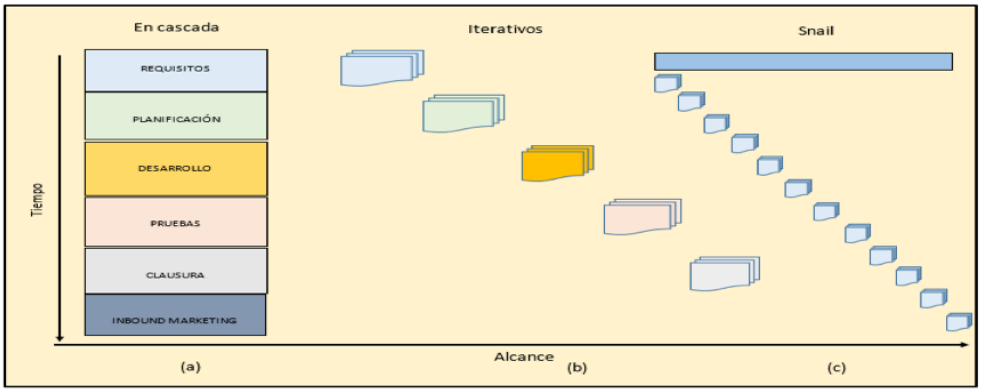

Ilustración 1: Evolución de los largos ciclos de desarrollo (a) a ciclos iterativos más cortos (b) y a la mezcla que hace SNAIL.

2.1.5. Proceso de SNAIL

Un proyecto de SNAIL tiene éxito cuando el cliente toma el precio de negocio a implementar basado en la habilidad que posee el equipo para establecer la funcionalidad que puede entregar a través del tiempo. Teniendo en cuenta estas características se establece la siguiente tabla en donde se definen las fases de la metodología y sus respectivas actividades que se debe llevar durante el proceso de desarrollo de software web.

Tabla 1: Fases y Actividades de la metodología SNAIL.

Fases	Actividades
Requisitos	1. Entorno de empresa
	2. Estudio de factibilidad
	3. Identificar actores o usuarios
	4. Identificar objetivos o requisitos
	5. Identificar requisitos funcionales y no funcionales
	6. Clasificar requisitos funcionales entorno a la funcionabilidad y mantenibilidad
	7. Validar requisitos
Planificación	1. Selección de requisitos o historias de usuarios
	2. Definir entregables
	3. Estimación de costos
	4. Velocidad del proyecto
	5. Escenarios de Bechmarking
Diseño	1. Diseño de bases de datos
	2. Diccionario de datos
	3. Diseño conceptual
	4. Diseño Navegacional
	5. Diseño de interfaz abstracta
Programación	1. Codificación
	2. Estándares
	3. Prueba unitaria
	4. Interconexión
	5. Integración
Pruebas	1. Análisis de benchmarking
	2. Pruebas de aceptación
Clausura	1. Presentar entregables
	2. Evaluación iteral del proyecto
Inbound marketing	1. Publicidad
	2. Branding

El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

1. El cliente establece el precio de negocio a implementar.
2. El programador mide el esfuerzo necesario para su implementación.
3. El cliente empieza a pedir que desea elaborar, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este modelo, tanto el cliente como el programador ganan experiencia. No se debe presionar al programador a realizar más trabajo del que ya se ha pautado, debido a que se podría perder la calidad en el software o no se cumplirán los plazos. Así mismo, el cliente tiene la obligación de manejar el modelo de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

Si bien el ciclo de vida de un proyecto SNAIL es muy dinámico, se puede separar en las siguientes Fases, representadas en este modelo:

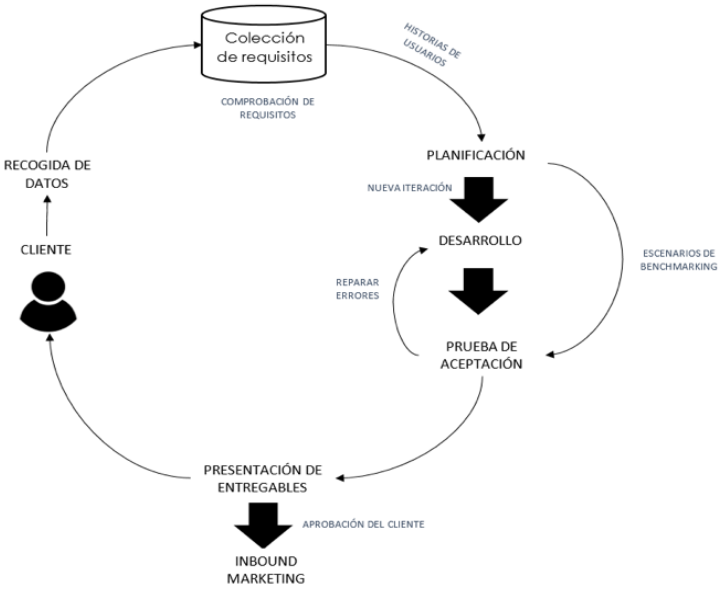

Ilustración 2: Fases del modelo SNAIL.

2.1.5.1. Fase I: Requisitos

En esta fase se definen todas las peticiones que se deben llevar a cabo durante la realización del proyecto del software: El entorno de empresa, realizando un estudio de factibilidad, identificando a los actores o usuarios (sus clases o perfiles).

También se analizan las necesidades de los usuarios finales del sistema para de esta manera determinar qué objetivos debe cubrir. A partir de esta fase nace una memoria llamada SRD (documento de especificación de requisitos), el cual posee la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos.

Es importante señalar que en esta etapa se debe detallar todo lo que el sistema necesita y requiere, debido a que serán estas especificaciones lo que seguirá en las siguientes etapas, sin permitir requerir nuevos resultados a mitad del proceso de elaboración del sistema, de ninguna manera.

2.1.5.2. Fase II: Planificación

En esta fase se proporciona un marco de trabajo que permite al gestor hacer estimaciones razonables de recursos, entregables, costos y planificación temporal. Estas estimaciones se hacen dentro de un marco de tiempo limitado al iniciar un proyecto de software, y tendrán que ser actualizadas constantemente a medida que avanza el proyecto. También, las estimaciones deberían definir los escenarios del mejor caso, y peor caso, de modo que los resultados del proyecto pueden limitarse.

2.1.5.3. Fase III: Diseño

En el diseño del sistema se reestructura y organiza el sistema en elementos que puedan ser desarrollados individualmente, aprovechando las ventajas del desarrollo en equipo.

Es conveniente distinguir entre diseño de alto nivel o arquitectónico y diseño detallado. El primero, tiene como objetivo establecer la estructura de la solución (una vez que la fase de análisis haya definido concretamente el problema) identificando grandes módulos y sus relaciones. Con esto se estructura la arquitectura de la solución elegida. El segundo define los algoritmos empleados y la organización del código para comenzar la implementación.

En cambio, en el diseño del programa se establece la fase donde se realizan los algoritmos necesarios para el cumplimiento de los requerimientos del usuario, así como también los análisis necesarios para saber qué herramientas usar en la etapa de Codificación, además del diseño de las bases de datos interfaz de usuario y diseño conceptual.

2.1.5.4. Fase IV: Programación

Esta es la fase en donde se transcribe el código fuente, haciendo uso de prototipos, así como de pruebas y ensayos para evitar tener errores y de tenerlos, lograr corregirlos a tiempo. Independientemente del lenguaje de programación que se maneje y su versión; se crean las bibliotecas y componentes que se reutilizarán dentro del mismo proyecto para hacer proceso mucho más rápido de la programación.

2.1.5.5. Fase V: Pruebas

Las partes que ya han sido programados, se juntan para estructurar el sistema y se comprueba que funciona correctamente y que cumple con los requisitos, antes de ser entregado al cliente.

2.1.5.6. Fase VI: Clausura

Es la última de las fases sino se toma en cuenta al Inbound Marketing (Que es opcional en esta metodología). Esta fase compone el proceso de gestión del mismo, y aplica tanto al proyecto en general como a cada una de las fases de su ciclo de vida. Teniendo así y de esta manera, tenemos un proyecto que se ejecuta en fases. Cada una de las fases debe incluir su proceso de aceptación y cierre, ajustado a sus características concretas.

2.1.5.7. Fase VII: Inbound Marketing

En esta fase (Opcional en la metodología) se realiza una estrategia que, basada en atraer clientes con contenido útil, relevante y agregando valor en cada una de las etapas del recorrido del comprador. Con la fase de Inbound marketing los clientes potenciales encuentran tu sistema a través de distintos canales como blogs, motores de búsqueda y redes sociales.

CAPÍTULO 3: FASE DE REQUISITOS

3.1. Fase de requisitos

Según Benet “Los requisitos son la especificación de lo que debe hacer el software, son descripciones del comportamiento, propiedades y restricciones del software que hay que desarrollar” [7].

- Los requisitos se pueden definir como objetivos a satisfacer mediante el proceso de construcción del software para que este sea exitoso y aceptable por el cliente.
- “Un requisito del software es una característica que debe exhibir para solucionar cierto problema del mundo real. Por lo tanto, un requisito del software es una característica que se debe exhibir por el software desarrollado o adaptado para solucionar un problema particular” [8].

Durante el proceso en el que el software es diseñado, desarrollado y finalizado, se necesita documentar y validar los requisitos de software, con el objetivo de dar al cliente un producto de software con éxito y que pueda completar sus necesidades de software. Estos requisitos son refinados y descritos de tal manera que, a la hora de crear el software, permita centrarse inequívocamente en cumplir sin variaciones o malentendidos las peticiones del cliente.

El tener requisitos incorrectos pueden causar que estos no puedan resultar en un software eficaz o satisfactorio, además de:

- Sobrecosto.
- Reproceso costoso.
- Mala calidad.
- Retraso en la entrega.
- Clientes descontentos.
- Miembros de equipo agotados y desmoralizados.

La importancia de tener requisitos de calidad radica en:

- Involucran del 10 al 15% del coste total del proyecto.
- Un error en los requisitos puede ser de 10 hasta 100 veces más costoso que un error en el código.
- Una equivocación en la etapa de requisitos se arrastra en las demás fases.

En el desarrollo de la obtención, refinamiento y especificación de los requisitos se debe realizar una estimación veraz del tiempo que se necesita para realizar o cumplir con el desarrollo del mismo, con el efecto de minimizar riesgos de fracaso y costos excesivos debidos a los requisitos mal planteados, por estas razones los requisitos deben tener las siguientes características:

- Ser una combinación compleja de los requisitos (necesidades) de diferentes personas (Stakeholders) que pertenecen a diferentes niveles de una organización y entorno en donde se operará el software.
- Deben ser verificables.
- Deben ser lo más claros que se pueda y cuantificables en medida de lo posible.

Dentro de los principales problemas que encuentra el desarrollo de software dentro de la ingeniería es que se basan en modelos o métodos que permiten la creación de un software sucesivo, implicando que, si existe un fallo en las primeras etapas, el resto de las mismas serían defectuosas provocando el efecto "dominó", para evitar este tipo de errores masivos en el producto de software, es imprescindible que se traten con mucha importancia los requisitos de usuario, sistema, etc.

"La correcta obtención de los requisitos es uno de los aspectos más críticos de un proyecto software, independientemente del tipo de proyecto que se trate, dado que una mala captura de los mismos es la causa de la mayor parte de los problemas que surgen a lo largo del ciclo de vida". [54]

Las principales dificultades que se presentan en el proceso de recolección de requisitos se pueden mencionar que los requerimientos:

- No reflejan las necesidades reales de los clientes
- Son inconsistentes y/o incompletos.
- Realizar cambios sobre los requisitos ya definidos es muy costoso.
- Pueden existir malentendidos entre los Stakeholders, y los ingenieros de software.
- Imprecisión de los requisitos, lo cual provoca que sean interpretados de diferentes formas por los Stakeholders.
- Frecuentemente no está clara la frontera entre requisitos y diseño.

3.1.1. Modelado de negocio

El modelado de negocios surge a partir de un conjunto de tareas en las cuales los desempeñan distintos roles de acción de acuerdo a las líneas de trabajo establecidas por la compañía, teniendo como atributo el ser dinámica en el análisis de entornos e interacción de dichas tareas que contienen o producen informaciones. Además, estos procesos del negocio están restringidos por las reglas de negocio que determinan las políticas y la estructura de la información de la organización [9].

El modelado del negocio tiene como objetivo principal el determinar y especificar los procesos de la organización y/o negocio, obteniendo información, actividades, roles y reglas del negocio implicadas. La especificación pretendida por el modelado es el centrarse como estos procesos interactúan con el sistema a desarrollar, siendo la cuestión única el saber qué es lo que hace dicho proceso al tomar papel con el sistema de informático y no el como lo hace, implicando que solo interesan las interacciones que se van a desempeñar con él.

3.1.1.1. Identificación de los procesos de negocio y los procesos de mantenimiento.

Al iniciar con el modelado de negocios el primer paso a desempeñar es el obtener el conjunto de procesos de negocios que realiza la organización para la cual se va a realizar el software, este conjunto debe ser adecuado, debido al gran impacto e importancia que acarrea, ya que al determinarlos estos proveen o establecen hasta donde llegará el modelado. Este conjunto de procesos son obtenibles mediante los objetivos a cumplir por la organización dentro del sistema [10].

Al obtener los objetivos del negocio es importante se tengan claros, debido a los fallos que pueden provocar para especificar sus procesos; una vez obtenidos según la complejidad de los procesos, estos serán divididos en nuevos sub-objetivos los cuales se refinaran en nuevos sub-objetivos, obteniendo una jerarquía que pueda llegar a ser comprensible por el equipo de trabajo, siendo suficiente hasta dos escalas de sub-objetivos. Para cada uno de los sub-objetivos de último nivel definimos un proceso del negocio cuyo cometido será alcanzar dicho objetivo, por último, estos procesos serán representados por casos de uso del negocio, los cuales en un principio serán descritos de forma textual. En esta etapa debemos distinguir entre procesos de negocio y procesos de mantenimiento de la organización. Estos últimos se detectan al encontrar en el sistema cierta actividad necesaria para mantener la Información de la organización. Esta actividad no puede estar encuadrada en ningún proceso de negocio [11].

3.1.1.2. Identificar los usuarios, departamentos o elementos de la organización implicados en el proceso de negocio.

En la identificación de los procesos de negocio, simultáneamente se pueden obtener los actores que intervienen en el mismo, los cuales serán denominados como rol, siendo este un agente que interactúa de forma externa a para llevar a cabo sus procesos de negocio.

Para representar los procesos de negocio se diseñan los diagramas de casos de uso del negocio donde cada caso de uso representa un proceso de negocio y cada actor representa un rol. Este diagrama permite mostrar los límites, contexto o entorno del sistema de información bajo estudio, donde sólo solo los roles externos al sistema serán visualizados, sin tomar en cuenta los internos dentro de la organización [12].

3.1.1.3. Descripción textual del proceso de negocio.

Se necesita determinar aquellos agentes (personas, departamentos, sistemas software, etc.) que están conectados directamente con el proceso de negocio. Debemos determinar también cuáles son las acciones (actividades) que realizan los actores dentro del proceso de negocio, y la información necesaria y producida por dichas actividades. El resultado de este paso es un listado de los actores que intervienen en el proceso y de las actividades que cada uno de ellos lleva a cabo.

3.1.1.4. Construir un diagrama de actividades que represente el proceso de negocio.

Para representar de forma detallada y rigurosa cada flujo de trabajo que se produce en el proceso se procederá a diseñar una serie de diagramas de flujo de actividades de cada proceso obtenido con anterioridad, lo cual nos permite tener una visión dinámica de cómo se desarrolla el mismo. Estos diagramas muestran las vías de actividades que ocupan cada rol, además de permitir el reconocimiento de la información que se requiere en esa actividad y rol, así como también la información que se produce en cada actividad y la sincronización requerida entre las diferentes actividades.

Es posible encontrar una actividad cuya complejidad sea tal que sea necesario describirla con un nuevo diagrama, siendo este un sub-objetivo en relación al objetivo principal, el cual está estrechamente relacionado con el proceso de negocio original [13].

Dentro de una organización se pueden identificar procesos de negocio que no requieran una interacción entre diferentes agentes, sino que todas las actividades son realizadas por un mismo agente. A tales situaciones les correspondería un diagrama de proceso con una única calle que incluiría el flujo de actividades realizado por un mismo rol.

3.1.1.5. Especificar las actividades que aparecen en el diagrama de actividades.

Las actividades que se obtienen dentro de los procesos de negocio son necesariamente específicas, esto para no caer en ambigüedades que permitan errores posteriores en el desarrollo, comprobar este tipo de fallos resulta importante debido a que son [13]:

a. Especificaciones de las actividades necesarias para lleva a cabo el proceso de negocio sujeto de estudio,

b. Necesarias en la identificación posterior de un primer conjunto de casos de uso del sistema y;

c. Trazables entre casos de uso del sistema y procesos de negocio.

La semántica de cada actividad será especificada mediante un contrato que incluirá el origen de la actividad (actividades que la preceden), agente, y pre y post condiciones de la actividad. Además, cada uno de estos contratos está recogiendo algunos de los tipos de reglas del negocio (precedencia entre actividades, pre y post condiciones). La plantilla que utilizaremos es la siguiente [14]:

Tabla 2: Plantilla para especificar el diagrama de actividad 1.

CAMPO	DESCRIPCIÓN
Contrato	Nombre de la actividad realizada por los actores
Origen	Actividad/es precedente/s
Agente	Actor que realiza la actividad
Pre-condición	Estado previo a la realización de la actividad
Post-condición	Estado posterior a la realización de la actividad
Caso de uso	Caso de uso con el que se corresponde la actividad. Este campo no se completará hasta que no se tengan descritos los casos de uso y podamos establecer la correspondencia.

3.1.1.6. Especificar las informaciones que fluyen en el diagrama de actividades.

En esta etapa se debe especificar de forma detallada aquella información que fluye dentro del diagrama de actividades, lo cual tendrá repercusiones en la construcción del modelo conceptual. Conviene documentar cada una de las informaciones encontradas, para así poder identificar sus atributos y restricciones de integridad [15].

El conjunto de todas estas especificaciones conforma el diccionario de informaciones. Este es muy útil por:

a. Son la especificación de las informaciones (datos) necesarios (generados o requeridos) para llevar a cabo las actividades del proceso de negocio sujeto de estudio.

b. Permitir la identificación posterior de un primer conjunto de clases del modelo conceptual del sistema.

c. Permitir la trazabilidad entre clases del sistema y clases del modelo conceptual. Además, cada una de estas especificaciones está recogiendo algunos de los tipos de reglas del negocio (integridad referencial, atributos calculados a partir de otros, como los precios totales delos pedidos, por ejemplo, etc.

Para describir las informaciones utilizaremos la siguiente plantilla:

Tabla 3: Plantilla para especificar el diagrama de actividad 2.

Nombre de la información	
CAMPO	**DESCRIPCIÓN**
Atributos	Listado de los atributos de la información
Restricciones	Restricciones sobre los atributos de la información referidas tanto al significado como al valor de los mismos.
Clase	Nombre que tendrá la clase con la que se modelará esta información. En principio no se indica nada, y sólo cuando se identifica la clase en el modelado de análisis y diseño se completa con el nombre de la clase.

El conjunto de estas plantillas conforma el diccionario de informaciones.

3.1.1.7. Reglas del negocio

Son el conjunto de restricciones de la organización a la hora de realizar una determinada actividad, e incluyen las restricciones asociadas a las informaciones (restricciones de integridad), actividades, así como también criterios, políticas, reglas, etc, dentro de la organización. Estas reglas de negocio quedan reflejadas explícitamente en el diccionario de actividades y en el diccionario de informaciones, creados en las subetapas anteriores. De manera que si alguna regla de negocio (restricción) no queda reflejada dentro de los diccionarios queda descrita textualmente para que se tenga constancia en el modelo del negocio, consiguiendo así recoger todas las reglas de negocio [16].

Ejemplo conceptual del modelado de negocio

Existen diferentes tipos de formatos para un modelado de negocios exitoso, dentro de la metodología "SNAIL" se ha planteado el siguiente con el fin de obtener de forma simple y detallada cada actor, proceso, subproceso, objetivo y sub-objetivo del negocio.

Actores del Negocio

[En esta sección se debe describir a cada uno de los actores del Negocio que participan en los Casos de Uso del Negocio, un actor del Negocio es un usuario del Negocio o cualquier entidad que interactúa con el Negocio.]

[Actor del Negocio 1]

[Descripción del Actor 1 del Negocio]

[Actor del Negocio 2]

[Descripción del Actor 2 del Negocio]

Casos de Uso del Negocio

Diagramas de Casos De Uso

[Aquí se presentan los Diagramas de Casos de Uso del Negocio, para mostrar la interacción entre los Actores y los Casos de Uso del Negocio.]

[Caso de Uso del Negocio 1]

Descripción

[Explicar brevemente el propósito del caso de uso del Negocio]

Flujo de eventos principal

[En esta sección se realiza una descripción textual del flujo del Negocio que representa el Caso de Uso. El flujo debe describir que hace el Negocio para proveer de valor a un actor del Negocio, pero no como hace el Negocio para resolver sus problemas. Las alternativas simples se pueden presentar dentro del texto del Caso de Uso. Si el flujo alternativo es más complejo, use una sección separada para describirlo.]

Flujos de eventos alternativos

[Flujo alternativo 1]

[En esta sección se describen las alternativas más complejas a las cuales se hacía referencia en el Flujo de eventos principal. Estos flujos alternativos representan la

conducta alternativa normalmente debida a excepciones que ocurren en el flujo principal. Ellos pueden ser necesarios para describir los eventos asociados con la conducta alternativa. Cuando finaliza el flujo alternativo, se retoman los eventos del flujo de eventos principal a menos que se establezca otra cosa.]

[Sub-flujo alternativo 1]

[Los flujos alternativos pueden dividirse en subsecciones si esto aporta claridad]

[Sub-flujo alternativo 2]

...

[Flujo alternativo 2]

...

Diagrama de Actividad

[En esta sección incluye un Diagrama de Actividad que modele en forma gráfica los flujos del Caso de Uso del Negocio descritos previamente. Es importante que se muestren los distintos condicionales que existen en el proceso que determinan que se siga un camino u otro en la ejecución del proceso, y cuales son todas las opciones de terminación para los flujos modelados.]

Requerimientos especiales

[En esta sección se especifican los requerimientos especiales que son específicos a un caso de uso del Negocio, que son las características del caso de uso del Negocio no cubiertas por los flujos descritos.]

[Requerimiento especial 1]

...

[Caso de Uso del Negocio 2]

...

Reglas del Negocio

[En esta sección se especifican las Reglas del Negocio que se deben tener en cuenta en la ejecución de los distintos procesos definidos. Las Reglas del Negocio son tipos de requerimientos sobre como el Negocio, incluyendo sus herramientas, debe funcionar.

Pueden ser leyes y regulaciones impuestas al Negocio entre otras, y pueden ser clasificadas de distinta forma según el Negocio, por ejemplo, pueden estar agrupadas por dominio, usuario o grupo de productos, aunque una clasificación más formal las define como Restricciones y Derivaciones.

Las Reglas del Negocio deben ser expresadas con un nivel de formalismo que permita

su automatización, una forma puede ser utilizando el Object Constraint Language (OCL) especificado en UML. De todas formas, es útil mostrar las Reglas de Negocio como notas de texto en los elementos de los distintos diagramas y modelos, por ejemplo, en los Diagramas de Actividad.]

[Regla del Negocio 1]

...

3.1.2. Estudio de factibilidad

El desarrollo de un proyecto de un proyecto de software exitoso requiere el uso de un estudio de la Factibilidad de sistemas, siendo una de las primeras etapas del desarrollo de un sistema informático el estudio incluirá los objetivos, alcances y restricciones sobre un sistema, además de un modelo lógico de alto nivel del sistema actual [17].

- Factibilidad Técnica: ¿Existe la tecnología necesaria? ¿Está al alcance de la mano?
- Factibilidad Económica: Relación Costo/Beneficio. ¿Se paga el costo?
- Factibilidad Operativa: ¿Cuáles son las capacidades organizacionales para sostener el sistema?

3.1.2.1. Esquema explicativo para el estudio de factibilidad

El desarrollo de un estudio de factibilidad permite a los jefes de proyectos, explorar orientar la toma de decisiones en la evaluación de un proyecto y dentro del ciclo del proyecto, por ello, “SNAIL” propone el siguiente modelo que permite realizar con efectividad el estudio, explorando los conceptos más básicos para su creación:

3.1.2.1.1. Objetivos:

- Determinar la factibilidad técnica, económica, operativa y jurídica (y de ser necesarias otras) del proyecto.
- Lograr el conocimiento general del problema y la estructura de los requerimientos de información, con el fin de estimar los recursos necesarios.
- Planear alternativas de desarrollo.
- Realizar planeación general de actividades.

3.1.2.1.2. ¿Para qué sirve el Estudio de Factibilidad?

- Para evitar desarrollar proyectos que no son factibles.
- Para planear los recursos.
- Para "aterrizar" al personal administrativo de sistemas, usuarios, auditores, etc. respecto a las expectativas reales del sistema.

3.1.2.1.3. ¿Quiénes participan en la etapa?

- Los usuarios: proporcionan los requerimientos del sistema.
- Los analistas, programadores, jefes de proyecto, director de sistemas.
- Personal de auditoría: asegura los controles y seguridad del sistema.

3.1.2.1.4. Actividades de la etapa

- Entender el proyecto.
- Establecer la duración y tamaño del proyecto.
- Determinar costos y beneficios.
- Determinar la factibilidad del sistema.
- Elaborar un documento con recomendaciones en el cual debe quedar claro si el proyecto es o no factible.

3.1.2.1.5. El estudio de factibilidad contiene:

- Definición organizada de los requerimientos de información.
- Recursos requeridos para el desarrollo del proyecto.
- Análisis de factibilidad.
- Alternativas de desarrollo.
- Cronograma de actividades.

3.1.2.1.6. Descripción de los principales componentes del Estudio de Factibilidad

3.1.2.1.6.1. Reconocimiento general del sistema

- Ubicación del sistema (Organización, departamento, sistema).
- Organización: Objeto social, tamaño, organigrama, ubicación geográfica, visión, misión.

- Sistema: Objetivos del sistema: debe reflejar la necesidad de satisfacer información requerida por el usuario.
- Delimitación o alcance del sistema (módulos a implementar): Define los límites hasta los cuales se va a expandir el proyecto. Resuelve las siguientes preguntas: ¿Qué hará el sistema? ¿Qué no hará el sistema?
- Beneficios que traerá el desarrollo del sistema.
- Restricciones o condiciones que tendrá el sistema: a nivel tecnológico o de funcionalidad.
- Definición del sistema (Narración):
- Objetivos: Definir características y necesidades de información.
- Identificar cada uno de los componentes, las entradas, las salidas y las relaciones entre cada uno de ellos.

3.1.2.1.6.2. Recursos requeridos

Objetivo: Definir las características y cantidades necesarias de talento humano, recursos técnicos y materiales, así como sus costos, necesarios para el desarrollo del proyecto [18].

3.1.2.1.6.3. Usuarios del sistema

Se identifican los usuarios que utilizarán el sistema, el cargo (dentro de la organización) y la ubicación geográfica de cada uno de ellos.

- Usuario primario: puede definir con adecuado criterio y autoridad las características fundamentales del sistema, dado que éste hace parte de su responsabilidad.
- Usuarios secundarios: son ellos quienes utilizan el sistema y por lo tanto no tienen una visión global de éste, pero brindan aportes importantes a nivel de detalle en los procesos más comúnmente utilizados por ellos, especialmente en interfaces con otras aplicaciones.

3.1.2.1.6.4. Beneficios esperados del proyecto

Es necesario que se tengan los beneficios esperados para el proyecto, con el fin de tener viabilidad, para esto se necesita identificar los problemas del sistema y los costos presentados. Si el sistema propuesto elimina el problema o reduce su costo, puede decirse que se tendrá un beneficio en la cantidad que en la actualidad representa dicho costo, dentro de los beneficios que se pueden obtener, podemos

clasificarlos dentro de los siguientes grupos [19]:

- Beneficios tangibles: son de fácil cuantificación, generalmente están relacionados con la reducción de recursos o talento humano.
- Beneficios intangibles: no son fácilmente cuantificables y están relacionados con elementos como el impacto sobre aspectos como Good Will o mejora en otros procesos de la organización.

Ej. de beneficios:

- Mejoras en la eficiencia del área bajo estudio.
- Reducción de personal.
- Reducción de futuras inversiones y costos.
- Disponibilidad del recurso humano.
- Mejoras en planeación, control y uso de recursos.
- Suministro oportuno de insumos para las operaciones.
- Cumplimiento de requerimientos gubernamentales.
- Toma acertada de decisiones.
- Disponibilidad de información apropiada.
- Aumento en la confiabilidad de la información.
- Mejor servicio al cliente externo e interno
- Logro de ventajas competitivas.
- Valor agregado a un producto de la compañía.

- Cuantificación de beneficios: Es necesario cuantificar en unidades monetarias los beneficios durante los años de VIDA ÚTIL del sistema, serán usados los requerimientos para el cálculo de la relación: Costo/Beneficio, además de la Variable Costo Sma. Actual (A), Costo Sma. Propuesto (B), lo que resultará en Beneficio (A-B) y Vida Útil Total.

3.1.2.1.6.5. Costos del proyecto

Los costos del proyecto pueden estar dados por la adquisición o utilización de equipos, personal, compra de software, costos en los procedimientos para la recopilación de información, preparación de documentación, mantenimiento de sistemas etc. Deben ser calculados para cada uno de los recursos en cada una de las etapas del ciclo de vida del sistema.

Requerimiento para el cálculo de la relación: Costo/Beneficio.

3.1.2.1.6.6. Análisis de alternativas de implementación

Es necesario evaluar cada una de las alternativas de implementación, eligiendo una de ellas y descartando una por una las demás alternativas. Deben sustentarse las razones de cada decisión.

- Mejoras al sistema actual: en esencia el sistema actual se mantiene con algunas mejoras.
- Adquisición de paquetes
- Desarrollo externo a la medida – Outsourcing
- Desarrollo interno a la medida: realizado por analistas de la organización (nosotros).

3.1.2.1.6.7. Análisis de factibilidad del sistema

El estudio de factibilidad se hace necesario cuando el desarrollo del sistema no tiene una justificación económica establecida, existe un alto riesgo tecnológico, operativo, jurídico o no se cuenta con una alternativa clara de implementación [20].

- Factibilidad económica
  - Se debe hacer una comparación de los costos de las posibles soluciones contra los beneficios que ofrecen, de acuerdo con lo documentado en los numerales anteriores.
  - Indicador: Relación Costo / Beneficio (<1 Factible, >1 No factible).
  - Debe hacerse para cada una de las alternativas de implementación.
- Factibilidad técnica

  Se debe hacer un análisis de la tecnología requerida para conseguir la funcionalidad y el rendimiento del sistema propuesto, cuál es el riesgo de desarrollo y cómo afectan éstos elementos el costo del proyecto. Además, se debe definir si el problema se puede resolver con los medios actualmente existentes y el grado de adaptación de la solución a la tecnología con la que cuenta la organización, para ello se plantean las siguientes interrogantes:

  - ¿Existe la tecnología requerida para el desarrollo del sistema?
  - ¿Puede la organización acceder a dicha tecnología?

- Factibilidad operacional

  Explica cómo hacer funcionar el sistema propuesto en las operaciones de la organización. Muestra cómo puede cambiar el sistema el entorno del usuario y las actividades que realiza.

  Comprende dos aspectos:

  – Impacto sobre los usuarios: ¿Cuenta el proyecto con respaldo y compromiso verdadero, en términos de actitud y asignación de recursos? ¿Consideran que el proyecto beneficia a la organización? ¿Está la alta dirección comprometida?

  – Impacto sobre los demás sistemas de la organización:

    ◊ ¿Cuál será el impacto del sistema sobre otros procesos de la organización?

    ◊ ¿Será positivo o Negativo?

1.3.2.1.6.8. Definir el impacto sobre los planes generales de la organización o sobre su planeación estratégica

Se debe determinar cómo contribuye el sistema a:

- El logro de los planes del departamento.
- El logro de los planes específicos de la compañía.
- El logro de la misión organizacional.
- El logro de la visión organizacional.

1.3.2.1.6.9. Cronograma

Se realiza con base en las etapas del ciclo de vida y bajo las restricciones de tiempo que siempre existen.

Ejemplo de contenido del estudio de factibilidad

1. Reconocimiento general del sistema

1.1.Ubicación de la organización

1.1.1. Nombre y objeto social

1.1.2. Visión

1.1.3. Misión

1.1.4. Políticas

1.1.5. Tamaño de la organización

1.1.6. Organigrama

1.1.7. Ubicación geográfica

1.2. Área o departamento

1.2.1. Organigrama

1.2.2. Factores críticos de éxito

1.2.3. Planes o proyectos

1.3. El sistema

1.3.1. Definición del sistema actual (Desc. Textual + Diag. Actividades)

1.3.2. Problemas del sistema actual

1.3.3. Delimitación o alcance del sistema a desarrollar

1.3.4. Identificación de procesos de la organización

1.3.5. Requerimientos del sistema a desarrollar

1.3.5.1. Requerimientos funcionales: ¿Casos de uso – Para qué sirve?- Qué debe hacer el sistema?

1.3.5.2. Lista de informes que debe generar

1.3.5.3. Entradas

1.3.5.4. Almacenamientos

1.3.5.5. Interfaces con otros sistemas

1.3.5.6. Requerimientos no funcionales (ilities)

1.3.6. Objetivos del sistema a desarrollar

1.3.6.1. Objetivo terminal

1.3.6.2. Objetivos específicos

2. Recursos requeridos por etapa

2.1. Talento humano

2.2. Recursos técnicos

2.3. Otros recursos

3. Usuarios del sistema

3.1. Usuario primario

3.2. Usuarios secundarios

4. Beneficios esperados del proyecto

4.1. Beneficios tangibles

4.2. Beneficios intangibles

5. Costos o inversión del proyecto – por etapa –

6. Análisis de alternativas de implementación

6.1. Mejoras al sistema actual

6.2. Adquisición de paquetes

6.3. Desarrollo externo a la medida – Outsourcing –

6.4. Desarrollo interno a la medida

7. Estudio de factibilidad

7.1. Factibilidad económica

7.2. Factibilidad técnica

7.3. Factibilidad operativa

8. Impacto sobre los planes de la organización o sobre su planeación estratégica.

8.1. Impacto sobre la visión

8.2. Impacto sobre la misión

8.3. Impacto sobre la planeación

9. Cronograma

10. Conclusiones

11. Recomendaciones

12. Glosario

13. Bibliografía

14. Anexos

3.1.3. Identificar actores o usuarios y requisitos

La obtención de datos dentro de la ingeniería de requisitos para refinarlos en requerimientos necesarios en el sistema informático, obliga al ingeniero ejercer un control base para la administración de los mismos; con la finalidad de mantener y cubrir las necesidades de forma puntual. Para la Administración de requerimientos se adquieren las siguientes fases [21]:

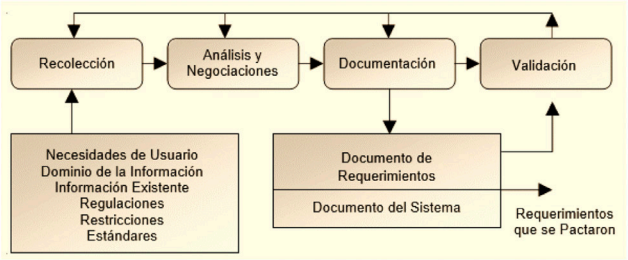

Ilustración 3: Proceso de la Ingeniería de Requerimientos. [21]

Estas actividades que conforman el Desarrollo de Requerimientos consisten en [21]:

- **Recolección (Elicitation):** Se entiende por recolección al proceso donde el cliente y el desarrollador articulan ideas, para llegar a obtener las necesidades de software del cliente para ser transformados en requisitos.
- **Análisis (Analysis):** Proceso por el cual los datos obtenidos en la recolección por medio de técnicas de obtención de datos empleados, se transforman en requisitos de software.
- **Especificación (Specification):** Desarrollo de documento estructurado que detalle de forma entendible cada uno de los requisitos.
- **Verificación (Verification):** Es el proceso para asegurar que la especificación de requerimientos de software sea acorde con los requerimientos del sistema, siguiendo una línea base conforme a estándares de documentación para los requisitos, siendo también una base para su próximo diseño y desarrollo.

3.1.3.1. FASE 1: Recolección

La recolección es la fase inicial en la cual se trata de descubrir los requerimientos e identificar los límites del sistema a través de la consulta a los participantes del sistema (Stakeholders). Dentro de los objetivos de esta fase se encuentran el entender el dominio de la aplicación, las necesidades del negocio, las restricciones del sistema, a los participantes del sistema y al problema en sí, para entender de manera inicial lo que se debe desarrollar [22]. Algunas de las técnicas y herramientas más importantes consideradas para la metodología “SNAIL” para llevar a cabo la recolección de requerimientos son:

3.1.3.1.1. Entrevistas

La entrevista es un método para descubrir hechos y opiniones que tienen los posibles usuarios y otros participantes dentro del sistema que se está desarrollando. Los errores y malentendidos pueden ser detectados y corregidos a través de este método, por lo cual resulta muy útil dentro de esta actividad de la ingeniería de requerimientos.

Es una forma de dialogo, formal o informal, donde el entrevistador busca la forma de obtener las respuestas necesarias para el desarrollo del software u opiniones acerca del mismo. La entrevista representa un gran reto tanto para el entrevistador como para el entrevistado, ya que debe haber un ambiente de confianza para obtener la información correcta [23].

Las entrevistas pueden ser clasificadas en dos grandes grupos.

- Las entrevistas cerradas, donde el entrevistador (ingeniero de requerimientos) prepara un conjunto de preguntas antes del encuentro con el entrevistado, y se buscan respuestas para las preguntas formuladas. Su estructura es parecida a una encuesta buscando obtener información más acertada y fiable, pero sin dar oportunidad al entrevistado de expresarse libremente.
- Las entrevistas abiertas, en las cuales no se preparan preguntas concretas, y, por el contrario, se discute con el entrevistado las expectativas que este tiene del sistema. En este tipo de entrevista apenas se hacen preguntas, se deja hablar al entrevistado. La ventaja de este tipo de entrevista es l flexibilidad con la cual cuenta el entrevistador.

No existe en realidad una delimitación entre los dos tipos de entrevistas en el momento de llevarlas a cabo. Pueden ser utilizadas de manera conjunta y no necesariamente exclusiva ni excluyente. Las diferencias podrían ser en cuanto a flexibilidad ya que la entrevista abierta posibilita una investigación más amplia.

La ventaja de este método es que permiten que el entrevistador obtenga una colección rica en información, que le puede ser útil al desarrollador. La desventaja que tiene este método, es que la información que se recolecta puede ser difícil de organizar y analizar, y diferentes participantes dentro del desarrollo del sistema pueden proveer información conflictiva y contradictoria, esto consecuencia de la gran cantidad de información que es obtenida durante la ejecución de este método.

Al poder realizar de manera frontal se puede contar con la ventaja de la información acertada y ser verificada por la persona encargada de la entrevista, pero su desventaja es el exceso de información si esta se realiza a más de una persona causando

confusión en la persona encargada del desarrollo del sistema además del tiempo que implica aplicar este método [24].

3.1.3.1.2. Casos de uso y/o escenarios

Los casos de uso describen interacciones entre los usuarios y el sistema, enfatizando en lo que el usuario necesita del sistema. Un caso de uso describe la posible secuencia de interacciones que se dan entre el sistema y uno o más actores como respuesta a un estímulo inicial por parte de alguno de los actores. De igual manera, debe ser incluida dentro de esta interacción, la descripción de las variantes y extensiones que el sistema debe soportar.

Considerado como una secuencia de transacciones describiendo las acciones a realizar por parte de los usuarios los cuales harán uso del sistema, los personajes o entidades que hacen uso del sistema se denominan actores [25].

Los casos de uso representan los requerimientos funcionales del software y pueden ser utilizados dentro de las primeras etapas del proceso de desarrollo. Así mismo, los casos de uso están escritos en lenguaje natural y son descripciones expresadas de manera informal. Las descripciones expresan lo que sucede desde el punto de vista del usuario. Los detalles de cómo el sistema debe funcionar internamente son irrelevantes al caso de uso.

Los casos de usos son empleados comúnmente para la captura de los requisitos funcionales. Cada caso de usos proporciona uno o más escenarios descritos en lenguaje natural proporcionados por los usuarios del sistema.

Por otra parte, los escenarios son ejemplos de sesiones de interacción entre el sistema y el usuario, donde un solo tipo de interacción entre los dos participantes es simulada y descrita. Los escenarios deben incluir una descripción del estado del sistema antes y después de la culminación del escenario, que actividades deben ser simultaneas, el flujo normal de los eventos y las excepciones a esos eventos.

Un escenario define situaciones en las que se puede encontrar un sistema, es decir define lo que puede hacer el sistema, considerando lo anterior descrito, un escenario puede tener diferentes formas de ocurrir dividido en acciones ya que estas acciones definen que camino debe tomar el escenario.

3.1.3.2. FASE 2: Análisis

El Análisis de Requerimientos de Software es el proceso en el cual se estudia las necesidades del usuario para obtener una definición de los requerimientos de software. Una Especificación de Requerimientos de Software es un documento en el cual se describe de manera clara y precisa cada uno de los requerimientos esenciales

(funcionalidad, rendimiento, restricciones de diseño y atributos de calidad) de un software y sus interfaces externas.

Parte fundamental del proceso de desarrollo de un sistema, trata de describir detalladamente los requerimientos de funcionalidad y calidad de servicio con los cuales constara el sistema en desarrollo, plasmándolos en un documento textual describiendo de forma concisa cada uno de dichos requerimientos [26]. Durante el análisis no se consideran descripciones específicas como requerimientos a menos que el cliente lo pida ya que están centrados en el problema a resolver.

Dentro de las actividades que se llevan a cabo en el análisis de requerimientos se encuentran: la descomposición de los requerimientos de alto nivel en requerimientos funcionales detallados, construcción de modelos gráficos de requerimientos, construcción de prototipos, priorizar los requerimientos, establecer atributos asociados con los requerimientos como puede ser su costo, o el beneficio que puede representar para el negocio, detectar y resolver conflictos entre requerimientos, descubrir el alcance del sistema y cómo interactúa en su ambiente, etc.

Durante la fase de análisis se subdivide en las tareas descritas anteriormente, considerada como una tarea fácil, lo cual resulta engañoso ya que al realizar mal esta tarea podría resultar fatal para el desarrollo del sistema, por lo cual se realizan diferentes métodos para el correcto participe del cliente y el analista, dentro de las prácticas principales para el análisis de requerimientos se encuentran:

3.1.3.2.1. JAD (Joint Application Development)

Esta práctica se basa en la creación de espacios que permitan celebrar sesiones o reuniones en donde los participantes y directos interesados dentro del desarrollo del proyecto buscan obtener o generar conocimiento alrededor del desarrollo que se va a llevar a cabo. En estas sesiones se trabaja bajo un enfoque común que permite el fácil entendimiento de los temas expuestos por parte de los invitados a la sesión (usualmente un enfoque de análisis estructurado), y se persiguen como propósito diferentes aspectos: definir niveles de detalle del proyecto, diseñar una solución, monitorear el proyecto, etc.

Las personas que pueden ser partícipes de estas reuniones pueden variar desde ejecutivos, líderes de proyectos, usuarios, expertos en el sistema y personal técnico externo. Este tipo de herramientas buscan obtener y maximizar el potencial de la cooperación y el trabajo en equipo entre los diferentes tipos de participantes [27].

Técnica de definición de requisitos y diseño de interfaces basada en reuniones entre el cliente, directivas y desarrolladores. Incluye enfoques para la mejora en la participación que tendrán los usuarios y agilizar el proceso de desarrollo del sistema.

Para sacar mayor beneficio al utilizar esta técnica, todos los partícipes deben implicarse al máximo para obtener resultados óptimos.

Dentro de las consideraciones que se deben tener en cuenta cuando se utiliza esta herramienta de trabajo es el nombrar un líder de las sesiones que impida que el rumbo de las sesiones se salga del propósito establecido de la sesión. Así mismo, la información que se obtiene en estas sesiones proviene de diferentes fuentes y representa los intereses de diferentes aspectos del sistema que se desea desarrollar, por lo tanto, es importante consignar esta información como base para la retroalimentación del proceso de ingeniería de requerimientos.

El propósito de estas sesiones es obtener, definir y revisar los requerimientos con los cuales constara el sistema. Al realizar esta técnica correctamente permitirá ver conflictos entre los requerimientos y eliminar aquellos que se consideren poco útiles.

3.1.3.2.2. Priorización de Requerimientos

Para mejorar la toma de decisiones en el momento de diseño y desarrollo, así como en otros aspectos de desarrollo del sistema es importante establecer un procedimiento que facilite esta importante actividad. Un potencial remedio para este dilema es la priorización de requerimientos, que permite manejar la situación descrita anteriormente. Esto permite controlar las decisiones que se realicen teniendo en cuenta a la fuente generadora de las necesidades.

La priorización ayuda a definir los requerimientos más importantes definiendo de esta manera los relevantes para el sistema y aquellos que son triviales, debido a la poca viabilidad de la implementación de todos estos requerimientos de una sola vez. En este proceso es de suma importancia la participación del cliente para lograr una correcta asignación de prioridades.

3.1.3.2.2.1. Modelos

La definición más general que se puede encontrar para la palabra modelo es:

- Representación en pequeño de alguna cosa.
- Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, como la evolución económica de un país, que se elabora para facilitar su comprensión y el estudio de su comportamiento.

En ingeniería de software, existen dos tipos de modelos básicos: el modelo conceptual y el modelo de comportamiento.

- Modelo conceptual: Es el utilizado en la especificación del sistema, representa los conceptos más significativos en el dominio del

problema…. Nos describe la parte estática del problema, es una fotografía del mundo real.

- Modelo de Comportamiento: Utilizado en la parte de diseño del sistema, define la parte dinámica, es decir, cuál debe ser el comportamiento en cada situación y la forma de proceder. Los diagramas de secuencia y de estados son parte de este modelo.

3.1.3.2.2.2. Características de un Buen Requerimiento

Las características de un requerimiento son sus propiedades principales. Un conjunto de requerimientos en estado de madurez, deben presentar una serie de características tanto individualmente como en grupo.

3.1.3.2.2.2.1. Características de la Descripción de un Requerimiento

Las características que tiene una buena descripción individual de un requerimiento, que lo diferencian de uno mal descrito, son [28]:

- Completo: Cada requerimiento debe describir de manera completa la funcionalidad que debe cumplir. Debe contener toda la información necesaria para que el desarrollador diseñe e implemente tal funcionalidad.
- Correcto: Cada requerimiento debe describir de manera precisa la funcionalidad que se debe construir. Un requerimiento correcto no debe entrar en conflicto con otro requerimiento. Sólo los usuarios más representativos del sistema pueden determinar de manera precisa si un requerimiento es correcto o no.
- Realizable: Debe ser posible implementar cada requerimiento de acuerdo a las capacidades y limitaciones del sistema y el medio que lo rodea. Para garantizar que no se determinen requerimientos no realizables, se recomienda contar con personal al interior del equipo de analistas de requerimientos que pueda establecer las limitaciones técnicas y de costos.
- Necesario: Cada requerimiento debe documentar algo que los clientes realmente necesiten, algo que sea para conformidad de un sistema externo con el que se tenga interacción, o para satisfacer un estándar. Para determinar si un requerimiento es necesario se debe determinar quién lo propuso, es decir, conocer su origen.
- Priorizable: Es importante asignar una prioridad para cada requerimiento que indique que tan esencial es el mismo para la realización del producto. Se pueden perder elementos de juicio para el desarrollo del sistema si se asigna el mismo grado de prioridad a

todos los requerimientos.

- No Ambiguo: Todos los lectores de un requerimiento deben llegar a una misma y consistente interpretación del mismo. El lenguaje usado en su definición, no debe causar confusiones al lector.
- Verificable: Un requerimiento es verificable cuando puede ser cuantificado de manera que permita hacer uso de los siguientes métodos de verificación: inspección, análisis, demostración o pruebas.

3.1.3.2.2.2.2. Características de la Especificación de Requerimientos

Así mismo, las características que debe poseer en conjunto una buena especificación de requerimientos son:

- Completa: Una especificación de requerimientos está completa si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.
- Consistente: Una especificación de requerimientos es consistente si no existen requerimientos que se contradigan.
- Modificable: Una especificación de requerimientos debe permitir ser revisada y mantener un historial de cambios hechos sobre cada requerimiento. Esto requiere que cada requerimiento sea etiquetado de manera única y expresado de manera separada de otros requerimientos para permitir referirse a él de manera no ambigua.
- Trazable: Cada requerimiento debe poder permitir trazar una línea del tiempo en la cual indique sus orígenes, y permita ser extendido a otras etapas del desarrollo del producto.

3.1.3.3. Fase 3: Documentación

El propósito de la documentación de requerimientos es el comunicar los requerimientos a los diferentes participantes del desarrollo de software. El documento de requerimientos de software es una herramienta que puede servir como base para la evaluación de los requerimientos en el producto, y para la evaluación misma del proceso que se llevó a cabo (evaluar las actividades de diseño, implementación, las pruebas realizadas y la verificación y validación de estas actividades).

Es necesario la elaboración de un documento que especifique los requisitos del sistema, este documento es compartido con todos los partícipes del proyecto teniendo como objetivo la evaluación del sistema y de los procesos que se llevan a cabo del modelo utilizado.

3.1.3.4. Fase 4: Verificación

Es la etapa final de desarrollo de requerimientos. Su objetivo es el verificar que todos los requerimientos que aparecen en el documento especificado describan y representen de manera clara, lo que se desea implementar para el sistema. Esto implica verificar que los requerimientos sean consistentes y que estén completos. Esta actividad representa un punto de control interno y externo; interno, porque se debe verificar internamente lo que se está haciendo, y externo, porque se debe validar con el cliente.

Es el proceso de comprobación que asegura que los requerimientos estén claros y representen lo que se va a plasmar en el sistema. Cada uno de estos requerimientos deben comprobarse con la documentación respectiva y ser validados con la debida autorización del cliente.

3.1.3.5. Herramientas, técnicas y software

Como se analizó en la sección anterior, existen diferentes herramientas que se utilizan para llevar a cabo las fases de la ingeniería de requerimientos. Sin embargo, estas herramientas no son restrictivas para una única actividad dentro de este gran proceso.

Las técnicas y herramientas descritas anteriormente son de completa ayuda para el correcto análisis del sistema, fomentando una gran ayuda para la persona encargada de realizar el respectivo análisis.

Tabla 4: Herramientas que se utilizan para las diferentes fases de la ingeniería de requerimientos.

Herramientas	Extracción	Análisis	Especificación	Validación
Entrevistas y cuestionarios	X			
Sistemas existentes	X	X		
Grabaciones de video y de audio	X	X		
Brainstrorming (lluvia de ideas)	X	X		
Arqueología de Documentos	X	X		
Observación	X			
Prototipo Throw Away (no funcional)	X	X	X	
Prototipo Evolutionary (funcional)	X		X	X
Análisis DOFA		X		
Cadena de Valor		X		
Modelo Conceptual		X	X	
Diagrama de Pescado	X	X	X	
Glosario	X	X	X	X
Diagrama de Actividad		X	X	
Casos de uso	X	X	X	X
Casa de Calidad o QFD				X
Checklist	X		X	

Fuente: [29]

También existen herramientas de software que facilitan la realización de cada una de esas actividades.

Tabla 5: Características de las herramientas de Administración de requerimientos más utilizadas.

CARACTERÍSTICAS	DOORS	RTM Workshop	Caliber - RM	RequisitePro
Permite la conversión de documentos fuente para cargar los requerimientos a una Base de Datos (BD)	X	X	X	X
Importa los requerimientos desde tablas de word hacia BD	X	X	X	
Incorpora objetos no textuales como hojas de trabajo de excel e imágenes dentro de BD	X	X	X	
Sincroniza un SRS con el contenido de BD		X		X
Define diferentes atributos para diferentes tipos de requerimientos y ajusta atributos para requeriemientos individuales	X	X	X	X
Define directivas para Requerimientos	X	X		
Notifica a los participantes afectados del proyecto vía e-mail acerca de los cambios en los requerimientos	X	X	X	
Define relaciones de trazabilidad y vínculos entre los requerimientos individuales y entre estos y otros elementos del sistema	X	X	X	X
Permite definie opciones de usabilidad para requerimientos no funcionales	X	X	X	X
Incluye elementos para el aprendizaje como tutoriales y proyectos de ejemplo	X	X	X	X
Se integra con otras herramientas como: prueba, diseño y administración de proyectos	X	X	X	X
Define usuarios y grupos, con sus respectivos permisos de acceso	X	X	X	X
Habilita discusiones de requerimientos		X	X	X
Incluye interfaces web para consultas a la BD, discusiones y tal vez la actualización de atributos de requerimientos	X	X	X	X
Incluye un sistema para desarrollos de sistemas basados en propuestas de cambios	X	X		
Incluye una completa documentación de manuales de usuario	X		X	X

Fuente: [29]

Estas herramientas están orientadas a soportar algunas de las actividades de la ingeniería de requerimientos, pero principalmente se orientan a permitir la administración de los requerimientos en relación a estas actividades. Por esta razón se les considera herramientas administrativas, y no herramientas que pertenezcan a alguna fase específica de la ingeniería de requerimientos.

Contar con la herramienta adecuada es preciso para el momento de realizar el análisis, aunque no todas desarrollan las mismas actividades unas pueden resultar más útiles que otras, pero no están orientadas en una fase especifica de la ingeniería de requerimientos. Su correcta utilización depende de la persona encargada del análisis, para minimizar los errores que pueden ocurrir durante la fase en la cual se utiliza dicha herramienta.

3.1.4. Identificar requisitos funcionales y no funcionales

3.1.4.1. Recolección de requerimientos

Las actividades que se realizan en la fase de recolección de requerimientos, son determinantes a la hora de realizar el análisis de los mismos, ya que constituyen la base para dicho análisis.

Para la fase de recolección de requerimientos, se determinan las restricciones del análisis dentro de un conjunto de actividades a realizar con características de ser abiertas y modulares; lo cual favorece el desarrollo de esta fase a través de actividades desarrolladas en etapas estándares. El conjunto de actividades que se debe llevar a cabo en esta fase son [28]:

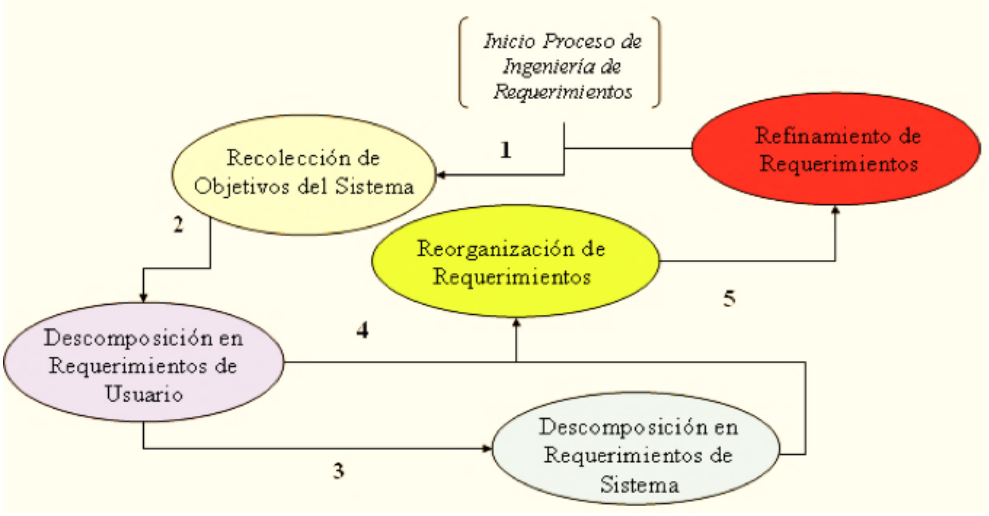

Ilustración 4: Modelo de proceso para Recolección de Requerimientos. [28]

Para facilitar la recolección de requerimientos el modelo se deriva en la simple tarea de recolectar con técnicas varias en nuevas que permiten que estos requisitos sean por ultimo refinados, con el fin de ser comprensibles para el equipo de desarrollo.

En el proceso de recolección de requerimientos no existe restricción en el cómo se deben obtener en empresas con un nivel de madurez alta, debido a que este tipo de

empresas contemplan sus propios procesos para realizarlo. Pero si se tratase de una empresa que aún no posee un nivel de madurez suficiente para tener este proceso constituido en los niveles 0 o 1 de la ilustración 4, se ve necesario el uso de técnicas que le ayuden en la obtención correcta de sus requisitos de sistema.

La metodología se basa en el modelo de recolección de requerimientos expuestos dentro del modelo previsto en la ilustración 4, así mismo expone algunas sugerencias de estrategias para llevar a cabo cada una de esas actividades.

3.1.4.1.1. Recolección de objetivos del sistema

Siendo la primera actividad dentro del desarrollo de un proyecto de software, el objetivo que se persigue dentro de la misma, es identificar los objetivos de negocio que se persiguen dentro de la empresa (objetivos de la empresa), para proceder con la extracción de requisitos.

Para a esta actividad "SNAIL" plantea las siguientes estrategias recomendadas para llevar a cabo esta actividad:

- Identificación de objetivos iniciales: Esta etapa trata la interacción directa con los Stakeholders para identificar los objetivos del sistema en base a documentos que pueden ser o no entregados por la organización. Este análisis se dará a partir de las entrevistas realizadas a los usuarios del sistema acciones que pueden identificarse como palabras especificas (verbos) dentro del documento; o, analizando cómo expresan sus requerimientos en forma de operaciones, procesos o diagramas de flujo, en cualquier documento (normalmente informal) que exista acerca del dominio del negocio donde la aplicación será desarrollada.
- Preguntas específicas: Es constituir un banco de preguntas específicas, con el fin de aplicar en una entrevista o cuestionario a los diferentes Stakeholders, de manera que se identifiquen los objetivos del sistema.
- Plantillas: esta estrategia propone el uso de una plantilla para la recolección de los objetivos del sistema. Esta, debe contener campos para ingresar información, como un número de identificación del objetivo, descripción del objetivo y otros definidos por el ingeniero de requerimientos.
- Estrategia lingüística: consiste en la formalización de un lenguaje sobre el cual se especifique y recolecte la información correspondiente a los objetivos del sistema.

3.1.4.1.2. Descomposición de los objetivos del sistema en requerimientos de usuario

Esta actividad persigue como objetivo el descomponer los objetivos previamente establecidos, en requerimientos más detallados (requerimientos de usuario), enfocados en la funcionalidad del negocio sobre el cual se está trabajando. Esta etapa es realizada para cada uno de los objetivos de negocio identificados en la fase anterior.

Para las empresas que no cuenten con un proceso maduro, y se encuentren en los niveles inferiores (nivel 0 y 1); se recomienda el uso de escenarios y/o casos de uso, las cuales se explican a continuación:

3.1.4.1.2.1. Casos de uso

3.1.4.1.2.1.1. Plantillas

Dentro de esta estrategia se establece un modelo para la creación de casos de uso, los cuales permitirán por medio de esta plantilla documentar los requisitos funcionales, esta utiliza una notación que permite implementar patrones lingüísticos a través una notación específica. Esta notación está compuesta así: Las palabras al interior de < > deben ser remplazadas de la manera apropiada, y las que se encuentran al interior de { } y separadas por coma, representan opciones excluyentes de manera que sólo una puede ser seleccionada [30].

Tabla 6: Plantilla para la descripción de requerimientos funcionales.

RF - <id>	<Nombre descriptivo>
Versión	<Número de versión actual> (<Fecha de versión actual>)
Autor	<Autor de versión actual> (<Compañía del autor>)
Fuente	<Fuente de versión actual> (<Organización fuente del requerimiento>)
Descripción	El sistema debe comportarse como se describe en la siguiente secuencia de interacciones cuando <evento disparador>
Alcance Y Nivel	<Que sistema es considerado caja negra en la perspectiva de diseño>
Precondición	<Lo que se espera del entorno para la realización del requerimiento>
Condición de Éxito	<Estado del entorno que indica que se realizó con éxito>
Condición de Fracaso	<Estado del entorno cuado el requerimiento es abortado o falla>
Actores	<Primario> {<Secundarios>}
Evento Disparador	<Evento disparador o trigger>

<table>
<tr><td rowspan="5">Secuencia Normal</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>...</td><td colspan="2">...</td></tr>
<tr><td>N</td><td colspan="2">{el { <actor>. Sistema} <acción</td></tr>
<tr><td rowspan="2"></td><td>n.1</td><td>{el { <actor>. Sistema} <acción ejecutada por actor/ sistema>,Pasos descritos en <caso de uso (RF-x)> es</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>...</td><td>...</td><td></td></tr>
<tr><td>Postcondición</td><td colspan="3"><Postcondición del caso de uso></td></tr>
<tr><td rowspan="3">Excepciones y Extensiones</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>P</td><td colspan="2">if <condición de excepción>, { el {</td></tr>
<tr><td>...</td><td colspan="2">...</td></tr>
<tr><td colspan="4">INFORMACIÓN RELACIONADA</td></tr>
<tr><td>Prioridad</td><td colspan="3"><grado de prioridad del requerimiento></td></tr>
<tr><td>Frecuencia</td><td colspan="3">Este caso de uso se espera ser ejecutado <número de veces> número de veces/ <unidad de tiempo></td></tr>
<tr><td rowspan="3">Desempeño</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>Q</td><td colspan="2">m segundos</td></tr>
<tr><td>...</td><td colspan="2">...</td></tr>
<tr><td>Canales hacia los Actores</td><td colspan="3"><Archivos, interactivo, base de datos, etc</td></tr>
<tr><td>Características Abiertas</td><td colspan="3"><Lista de carácterísticas que pueden afectar las deciciones sobre el caso de uso></td></tr>
<tr><td>Superiores</td><td colspan="3"><Lista de requerimientos que incluyen este requerimiento></td></tr>
<tr><td>Subordinados</td><td colspan="3"><Vínculos hacia subrequerimientos></td></tr>
<tr><td>Comentarios</td><td colspan="3"><comentarios adicionales acerca del requerimiento</td></tr>
</table>

Fuente: [31]

La correspondiente definición de cada uno de los campos de esta plantilla es:

- ***Versión***: De acuerdo a las recomendaciones de la IEEE, y de compañías líderes en procesos de software como Rational, un requerimiento debe permitir manejar sus distintas versiones de manera que se pueda analizar la evolución del mismo a través del tiempo. Para todo requerimiento, este campo debe estar ocupado por la versión actual, con su correspondiente número y fecha.

- ***Autor, Fuente***: Contiene el nombre de la organización y el autor del requerimiento.
- ***Propósito***: Contiene la descripción del porqué el requerimiento consignado es necesario para alcanzar los objetivos del negocio79.
- ***Descripción***: Para los requerimientos funcionales, esta plantilla contiene un patrón lingüístico que indica que debe ser llenado y los eventos que disparan el requerimiento.
- ***Alcance y Nivel***: Cuál sistema debe ser considerado como caja negra para este requerimiento.
- ***Precondición***: Se ingresan las condiciones necesarias que se deben tener para llevar a cabo el requerimiento que se está describiendo.
- ***Condición de Éxito***: Condición que indica si la ejecución del requerimiento fue exitosa.
- ***Condición de Fracaso***: Condición que indica que la ejecución del requerimiento fue abortada o falló.
- ***Actores***: Actor primario, y/o lista de actores secundarios del requerimiento.
- ***Evento Disparador***: Evento que dispara la realización del requerimiento.
- ***Secuencia Ordinaria***: Aquí se deposita la secuencia de interacciones que tiene el usuario con el sistema, en orden de llevar a cabo una funcionalidad u operación. Los pasos que se llevan a cabo en pueden a su vez contener sub-pasos o secuencias anidadas, asumiendo que sólo una de estas se puede llevar a cabo.
- ***Post-condiciones***: Son los estados a los cuales se debe llegar o los resultados que se deben obtener después de ejecutar la funcionalidad descrita en el requerimiento.
- ***Excepciones Y Extensiones***: Durante la interacción descrita en la secuencia ordinaria se pueden presentar excepciones o extensiones condicionales, debido a flujos alternativos de dicha interacción entre el usuario y el sistema. Es este campo se especifica la secuencia que se tomaría si se presenta la excepción o se indica si la funcionalidad descrita en el requerimiento se aborta.
- ***Prioridad***: Indica que tan importante es el requerimiento, que grado de prioridad tiene el requerimiento para los usuarios y clientes80.
- ***Frecuencia***: Frecuencia con la que se lleva a cabo la ejecución del requerimiento en un intervalo de tiempo. Se indica la tolerancia a fallos que tiene el requerimiento de acuerdo al número de eventos

que se utiliza el requerimiento y el tiempo que el mismo debe estar disponible.

- ***Desempeño***: Aquí se especifica un tiempo de tolerancia respecto al tiempo de respuesta del sistema para alguno o todos los pasos de la secuencia ordinaria.
- ***Canales hacia los actores***: Canales como archivos, interactivo, base de datos a través de los cuales se expresa el resultado de la ejecución del requerimiento
- ***Características Abiertas***: Lista de características que pueden afectar las decisiones sobre el caso de uso
- ***Superiores***: Lista de requerimientos que incluyen este requerimiento.
- ***Subordinados***: Vínculos hacia sub-requerimientos.
- ***Comentarios***: En éste campo se ingresa toda la información que no se pudo agregar a ninguno de los campos.

Así mismo, se debe puede tener una plantilla para la descripción de requerimientos no funcionales:

Tabla 7: Plantilla para la descripción de requerimientos no funcionales.

RN-#	
Versión	
Autor	
Fuente	
Descripción	
Prioridad	
Comentarios	

Fuente: [32]

- Gráficas y Diagramas: Los casos de Uso son construidos conforme a la notación estándar de UML, donde las descripciones de los casos de uso permiten múltiples actores, existiendo una visión única del caso para cada actor envuelto en él.
- Estrategia Orientada a Actores: propone identificar como primer paso a los actores del sistema, como medio para identificar los casos de uso.

3.1.4.1.2.1.2. Escenarios

Para la especificación de escenarios, se debe establecer un conjunto de reglas que permitan definir el contenido de los mismos, ya sea estableciendo una platilla como en los casos de uso.

- Recolección de requerimientos funcionales: se proponen dos estrategias para la recolección de este tipo de requerimientos.
- Refinamiento: se aplica sobre los casos de uso, de manera que se descomponen originando Requerimientos Funcionales de Sistema.
- Composición: permite identificar los objetivos complementarios de un requerimiento particular, y ayuda a identificar un paquete de servicios para un sistema determinado.

3.1.4.1.3. Reorganización de requerimientos

En esta actividad se plantea establecer una relación de seguimiento en los requerimientos, proveyendo de forma simple como estas deben ser cumplidas.

3.1.4.1.4. Refinamiento De Requerimientos

Dentro de esta etapa se pretende de forma eficiente el minimizar los riesgos-errores que se puedan aun conservar dentro de los requisitos, para esto se pretende seguir un proceso donde las siguientes actividades sean analizadas: eliminar duplicidad en los requisitos, unificar sinónimos, descubrir y eliminar inconsistencias, redundancias y requisitos incompletos, etc. Siendo SNAIL una metodología versátil, este proceso puede ser manual, o poseer algún nivel de automatización, sin caer en restricciones de desarrollo de la etapa, dando total libertad a la empresa de emplear los recursos o técnicas que cataloguen como necesarias de emplear.

3.1.5. Analizar los requerimientos

El proceso conocido como el Análisis de los Requerimientos es de los más importantes dentro de cualquier tipo de proyecto que necesite obtener requisitos del cliente, dentro este proceso existen consideraciones iniciales, las cuales son necesarias para el desarrollo de un concepto que cubra ampliamente las características indispensables para llevar a cabo un correcto y completo proceso de análisis de requerimientos, usados dentro de las organizaciones se encuentran los siguientes factores [33]:

- Identificar y definir los requerimientos de software.
- Manipular estos requerimientos como ítems de configuración y administración, así como se realiza en otros productos del ciclo de vida de un proceso de desarrollo.
- Detectar y resolver conflictos entre requerimientos.

- Descubrir los límites del software y cómo este debe interactuar con su ambiente.
- Clasificar los requerimientos, en módulos, de acuerdo a su funcionalidad.
- Describir al máximo las características y atributos de los requerimientos.

Para la etapa se prevé el establecer un conjunto de actividades estándares que se deben realizar para el desarrollo del software, teniendo un enfoque de modelo transparente y flexible a los procesos ya establecidos dentro de las empresas, sin tener en cuenta el enfoque o perspectiva del mismo, es decir que se puede usar el mismo modelo para cualquier tipo de análisis de requerimientos dentro de diferentes proyectos.

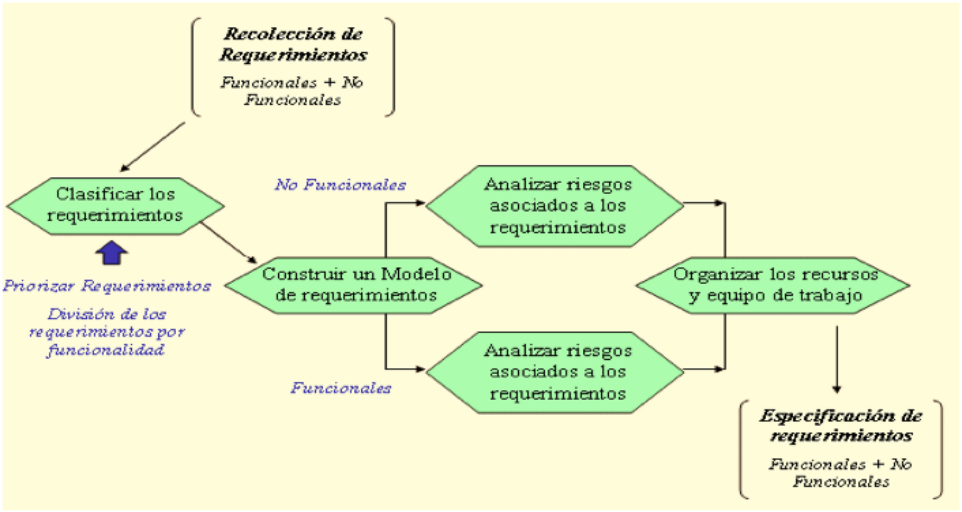

Ilustración 5: Modelo de proceso para análisis de requerimientos [33].

El modelo presentado tiene gran aceptación dentro de la metodología SNAII, ya que éste está orientado no solo a un tipo único de requerimiento, sino más bien se puede adaptar para cualquier tipo de requerimiento funcional y no funcional, así mismo es versátil dentro de su utilización, permitiendo que se pueda usar no solo en un tipo de proyecto, sino que en cambio se basa en satisfacer cualquier proyecto que implique el proceso de especificación de requisitos.

Cada etapa dentro del modelo está enfocada al análisis de los requerimientos necesarios para obtener una buena especificación de los requerimientos, las cuales son descritas a continuación [33]:

- Clasificar los requerimientos: La clasificación de los requerimientos de un proyecto, representa el dividir los requerimientos por módulos

del desarrollo, asociados por funcionalidad, y diferenciados por tipo, ya sea funcional o no funcional. También es necesario el establecer una medida de priorización de cada uno de los requerimientos, de manera que se facilite la toma de decisiones de desarrollo de los mismos.

- Construir un modelo de requerimientos: esta actividad se refiere a la creación de una forma para representar los requerimientos, sus características, propiedades, así como también escoger la perspectiva a través de la cual se analizarán los requerimientos, y cómo se llevarán a cabo las actividades y tareas restantes del análisis de requerimientos.
- Analizar riesgos asociados a los requerimientos: En esta etapa se plantea el analizar los riesgos necesarios para verificar el desarrollo del sistema, antes de empezar a desarrollarlo. Estos riesgos pueden darse por los siguientes aspectos: costo de desarrollo, costo no desarrollo, tiempo de desarrollo, dificultad de implementación, dependencia de otra funcionalidad, etc. Estos diferentes aspectos deben ser evaluados de manera que se pueda asociar un riesgo determinado.

La metodología SNAIL recomienda la utilización de análisis de riesgos para los requisitos que han sido obtenidos, pero de forma separada, debido a que estos dos tipos de requerimientos poseen características, especificaciones y efectos sobre el sistema, diferentes entre sí.

- Organizar los recursos y equipos de trabajo: La etapa se centra en repartir conforme al equipo de trabajo que haya sido designado a cumplir una tarea los recursos que se van a invertir en el desarrollo de los mismos (tiempo, dinero, desarrolladores).

3.1.6. Validar requisitos

Los requisitos son fundamentales para el éxito del producto final. Para satisfacer las necesidades del usuario, primero el equipo de proyecto debe enfocarse en el problema, es decir definir qué es lo que se desea construir y asegurar qué es necesario; aunque las pruebas del software no se ejecutan durante el desarrollo la realización de pruebas conceptuales ayudará a descubrir la existencia de requisitos defectuosos, sin claridad o incompletos.

En esta etapa se tiene mucha importancia a la validación de los requisitos conforme se vayan obteniendo, dado que si no se procede con algún tipo de validación y verificación, estos requisitos probablemente no cumplan concretamente con las necesidades del cliente [34].

Para entender lo que se refiere a la validación de requisitos, se definirá como el proceso en el que los requisitos son comprobados, orientándolos a la visión del cliente para poder tener la garantía total de satisfacción en las necesidades. Debido a esto se considera de las etapas de mayor importancia, dado que si se ha tenido una especificación de requisitos se corre el riesgo de obtener problemas y aumento de riesgos para que el proyecto falle.

Dichos aspectos tendrán impacto directo en el desarrollo, la finalización del producto, produciendo mayor o menor costo para el equipo de desarrollo, así como también en los tiempos de entrega. La validación de requisitos generalmente se realiza cuando el documento de especificación de requisitos tenga su primera visión general completa.

De esta manera es de suma importancia que al terminar la etapa de modelado de requerimientos se realice el proceso de comprobación realizando una comparación de las conversaciones iniciales que se realizaron a las especificaciones que se obtuvieron [32], para este proceso "SNAIL" propone la siguiente técnica, debido a su simplicidad y agilidad en torno a la posible corrección del producto con los requerimientos:

3.1.6.1. Revisión de pares

La revisión por pares es un método sencillo de verificación para un proyecto de software, debido a que solo se necesitan un conjunto de Stakeholders que se designaran para seguir un conjunto de reuniones, donde comprobaran si los requisitos han sido cumplidos de forma exitosa, debido a su sencillez, la metodología SNAIL recomienda el uso de esta técnica de verificación siguiendo estándares para la creación de equipos de trabajo y dado que esta planea iteraciones ágiles [35].

Para la revisión de pares, se plantea usar una sub-técnica o derivación que se considera más formal, siendo las inspecciones se pueden incorporar en las fases de: Planificación, Reunión de Información, Preparación de inspección individual, Reunión de Inspección, Seguimiento, Análisis causal (para determinar la causa de los defectos y decidir la forma de prevenirlos en un trabajo futuro), inspecciones sobre roles formales y herramientas de inspección.

Para realizar esta revisión se debe:

- Proporcionar un entorno de aprendizaje en el que los interesados puedan entender mejor los requisitos de los requisitos y dominio del negocio.
- Obligar a los analistas a centrar los esfuerzos de validación en las partes en que los requisitos tengan mayor riesgo y ambigüedad.
- Definir las expectativas de calidad para las necesidades a través de

la creación de listas de comprobación o revisiones de inspección.

- Identificar los requisitos de las oportunidades de mejora de procesos

Además de lo anterior se debe añadir los requisitos que van a ser comprobados y de qué forma lo van a realizar; identificar que Stakeholders serán los designados para realizar dichas comprobaciones, planificar la revisión a través de la organización de reuniones para las revisiones y finalmente se deben preparar las revisiones de las reuniones usando listas de chequeo.

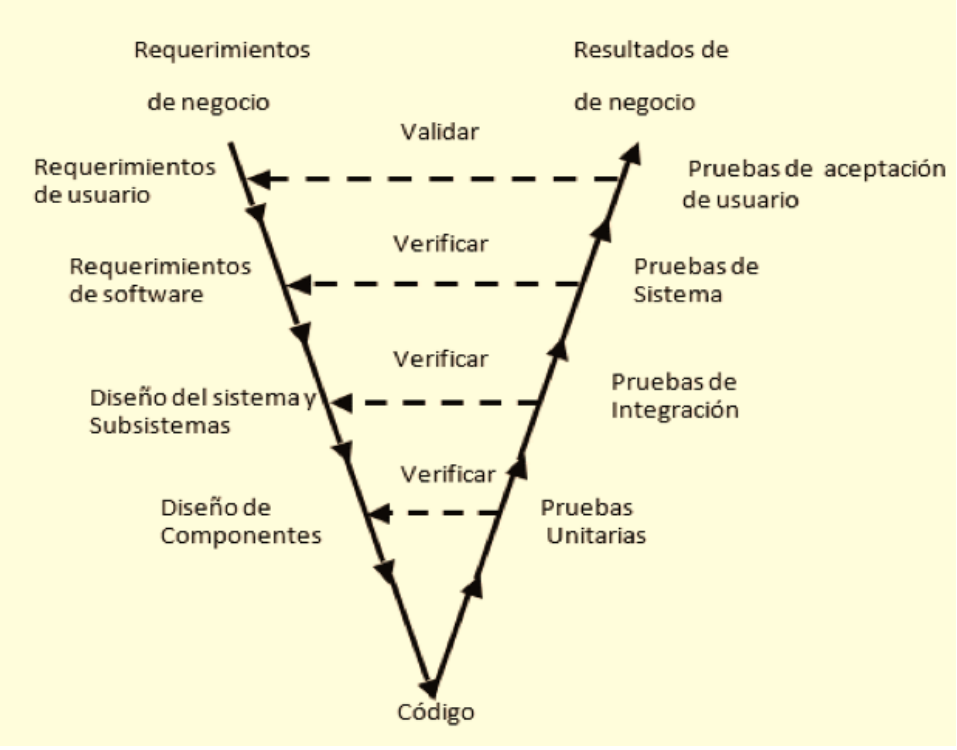

Ilustración 6: Proceso de verificar y validar requerimientos. [8]

Una vez se tienen correctamente obtenidos los requisitos, el equipo desarrollo procederá a comprobar si estos realmente cumplen con las necesidades, pero, solo desde el punto de vista del equipo de proyecto, la verificación consiste en cómo estos luego de ser desarrolladas satisface las necesidades del cliente, en torno a su conformidad dentro de lo que el haya necesitado.

Conclusión:

Este capítulo se ha desarrollado de tal forma que el proceso expuesto por la metodología SNAIL sea altamente fiable para que el desarrollador o equipo de desarrollo de proyectos, cree proyectos exitosos, dado que la fase de requerimientos es considerada la primordial dentro del desarrollo de todo proyecto, el énfasis especial en esta etapa es primordial.

Los requisitos de software dentro de la metodología SNAIL llevan procesos altamente versátiles, y fases como la obtención y especificación de los mismos no están sujetos a reglas estrictas, debido a que pueden existir empresas que ya cuenten con procesos para obtener, analizar y especificar los requerimientos. Por otro lado, esta nueva metodología se centra en empresas que no llevan un nivel de madurez apropiado para tratar con sus requisitos de software, dando pautas eficientes y necesarias para que todo grupo de desarrollo pueda realizar un proceso de verificación y validación de requisitos para que el proyecto no resulte en problemas comunes como son: incremento en los costos, insatisfacción de las necesidades, redundancia, etc.

CAPÍTULO 4: FASE DE PLANIFICACIÓN

4.1. Fase de planificación

La planificación se trata de una fase corta, donde se establece el orden en que se deberán implementar las historias de usuario, y, las respectivas entregas, esto mediante el acuerdo conjunto de los clientes, los gerentes y el grupo de desarrolladores.

Esta fase se basa principalmente en la ejecución de una o varias reuniones grupales de planificación. El resultado de esta fase es un Plan de Entregas.

4.1.1. Selección de requisitos o historias de usuario

Los documentos de especificación funcional y los casos de uso son reemplazados por las llamadas Historias de Usuario, las cuales son escritas directamente por el cliente, donde mediante sus propias palabras realizará descripciones poco detalladas de lo que el sistema informático debe cumplir.

Claro está que el nivel de detalle solicitado es la diferencia más significativa entre estas historias de usuario y los tradicionales documentos de especificación funcional. Debe existir un detalle mínimo en las historias de usuario con el fin de garantizar a los programadores una estimación poco riesgosa del tiempo que se invertirá en el desarrollo de dichas historias.

En la actividad de la codificación, los desarrolladores obtendrán los detalles necesarios mediante la comunicación directa con el cliente. Se debe cumplir con la programación de las historias de usuario en un tiempo mínimo de una semana y un máximo de tres semanas. En caso de que la estimación supere a tres semanas, la historia de usuario debe ser dividida en dos o más historias dependiendo el tiempo de la misma, en el caso de que la estimación sea menor de una semana, se debe complementar con otra historia.

4.1.2. Definir entregables

Mediante una reunión que incluya la participación de todos los actores del proyecto (gerentes, clientes, desarrolladores, etc.) se debe crear el cronograma de entregas donde se establecerá las agrupaciones de historias de usuarios que conforman una entrega y el orden que se deben realizar.

En esta actividad de acuerdo a las prioridades del cliente, él mismo ordenará y agrupará las historias de usuario que complementan una entrega. El cronograma de entregas se proyecta en base a las estimaciones de tiempos de desarrollo elaboradas por los desarrolladores. Es conveniente realizar nuevamente una reunión con los

actores del proyecto después de algunas iteraciones, con el objetivo de evaluar el plan de entregas y ajustarlo si el caso lo requiere.

En no más de tres meses se debe obtener una entrega. Como se había especificado las estimaciones de tiempo asociado a la implementación de las historias de usuario es establecida por los programadores utilizando como medida el punto, el cual equivale a una semana ideal de programación. Las historias de usuario se estiman de manera general entre de 1 a 3 puntos.

4.1.3. Estimación de costos

Independientemente del tamaño de un proyecto de desarrollo de software, una correcta estimación de su costo permitirá resolver problemas relacionados con el esfuerzo y tiempo invertido en su elaboración. En este apartado se analizan tres modelos de estimación de costos: modelo basado en líneas de código fuente (SLOC), modelo no-SLOC, y el modelo basado en puntos de casos de uso, con la finalidad de establecer sus ventajas de acuerdo a la cuatificabilidad y objetividad en el diseño de software a la medida.

El método de estimación más oportuno depende de los aspectos valorados en el análisis de un sistema de información, es decir el lenguaje de programación utilizado, documentación UML y el sueldo remunerado por experiencia y lenguaje utilizado.

4.1.3.1. Modelo basado en SLOC

En este modelo, para definir el tamaño de un producto de software se utiliza las líneas de código fuente. El modelo aquí estudiado es el Modelo Constructivo de Costos (COCOMO, Constructive Cost Model). COCOMO permite estimar la duración de un proyecto y el esfuerzo necesario para su elaboración medido en personas-mes. Por tanto, COCOMO divide los proyectos de software en tres tipos de acuerdo a su tamaño: modo orgánico, semi-acoplado y acoplado.

Ecuación para calcular el esfuerzo:

$$Ei=a*(KLOCS)^b$$

Ecuación para calcular el tiempo de desarrollo del proyecto:

$$Td=c*(Ei)^d$$

Donde:

Ei es el esfuerzo, medido en meses-hombre.

a, b, c y d: son valores que dependen del tipo de proyecto.

Td es el tiempo de desarrollo requerido en meses.

KLOCS: es el valor en miles de líneas de código.

Una de las ventajas más importantes de este modelo es que puede aplicarse en las distintas fases del ciclo de vida y en cualquier organización, además utiliza manejadores de costos que permiten al estimador analizar el impacto de otros factores que afectan en el costo del proyecto, entre estos factores puede estar presiones de tiempo, requisitos de desarrollo y tamaño. Por otro lado, la principal desventaja del modelo es que hace uso de datos históricos, como archivos de código fuente que ya no se utiliza, que no siempre están disponibles y es extremadamente vulnerable a la clasificación del modelo de desarrollo, ya sea programación estructurada o programación orientada a objetos.

4.1.3.2. Modelo no basado en SLOC

Este modelo es una opción para reemplazar al modelo basado en SLOCS y se basa principalmente en preguntas o transacciones de entrada. Entre estos métodos, el método de puntos de función (FPA, Function Point Analysis), se determina como un método estándar para medir el desarrollo de software desde el punto de vista del usuario.

Su función está dada a través de la asignación de puntos para así identificar los componentes del sistema en términos de transacciones y grupos de datos lógicos que son más importantes para el usuario.

Los puntos de función se encargan de medir el tamaño de una aplicación ya sea planificada o existente, además se pueden implementar para medir el tamaño de las modificaciones de una aplicación existente. En el caso de que las modificaciones se encuentren en los requerimientos funcionales del usuario o el diseño este culminado. Se debe seguir el siguiente proceso:

1. Establecer el tipo de conteo.

   Se puede tomar en cuenta tres tipos de conteo de puntos de función los cuales pueden ser para proyectos en desarrollo, mantenimiento y para una aplicación desarrollada.

2. Determinar el alcance de medición y los límites de la aplicación.

   El alcance se determina con la identificación de los sistemas, aplicaciones o subconjuntos de una aplicación a ser medidos. El límite de la aplicación está entre la aplicación medida y las aplicaciones externas al dominio del usuario.

3. Conteo de las funciones de datos.

   Contabilizar la capacidad de almacenamiento de los datos los cuales representan la funcionalidad que cumplen con los requerimientos de datos tanto internos como externos.

Se identifican dos tipos de funciones de datos los cuales son:

ILF (Archivo Lógico Interno): conjunto relacionado de datos internos determinados por el usuario, su objetivo principal es almacenar datos sostenidos a través de una transacción.

EIF (Archivo Lógico Externo): conjunto relacionado de datos externos referenciados, pero no sostenidos por alguna transacción dentro del conteo.

A continuación, se presenta el procedimiento a seguir:

- Identificar los archivos.
- Establecer a cada uno un tipo de función de datos, ILF o EIF
- Determinar la cantidad de DET (Data Element Type) el cual un campo único no repetitivo identificable por el usuario y RET (Record Element Type) que se trata de un grupo de campos en un archivo, identificable por el usuario.
- Establecer un valor de complejidad a cada uno ya sea baja, media o alta esto en relación de la cantidad de DET y RET como se puede observar en la siguiente tabla.

Tabla 8: Nivel de complejidad para los ILF y EIF.

	Número de campos			
Número de registros	***Para ILF/EIF***	***1 a 19 DET***	***20 a 50 DET***	***51 o más DET***
	1 RET	Baja	Baja	Media
	2 a 5 RET	Baja	Media	Alta
	6 o más RET	Media	Alta	Alta

4. Conteo de las funciones transaccionales.

Para contabilizar la capacidad de realizar operaciones, se dispone de tres tipos de funciones transaccionales:

Entrada Externa (EI): el objetivo principal es mantener uno o más archivos lógicos internos.

Salida Externa (EO): el objetivo principal es mostrar información al usuario a través un proceso lógico distinto al de sólo recuperar datos.

Consulta externa (EQ): el objetivo principal es mostrar información al usuario leída de uno o más grupos de datos.

A continuación, se presenta el procedimiento para el conteo de las funciones transaccionales:

- Determinar las transacciones.
- Establecer a cada una un tipo de funciones transaccionales EI, EO o EQ.
- Determinar la cantidad de DET y FTR (File Type Referenced) el cual es un tipo de archivo al que se hace referencia en una transacción, tiene que ser un ILF o EIF.
- Establecer un valor de complejidad a cada una ya sea baja, media, alta esto en relación de la cantidad de DET y FTR como se puede observar en la siguiente tabla.

Tabla 9: Nivel de complejidad para EI, EO y EQ.

EI/EO/EQ	1 a 4 DET	5 a 15 DET	16 o más DET
0 a 1 FTR	Baja	Baja	Media
2 FRT	Baja	Media	Alta
3 o más FTR	Media	Alta	Alta

5. Establecer los puntos de función sin ajustar (PFSA).

   Adicionar el número de componentes de cada tipo en función a la complejidad y basarse en la siguiente tabla para obtener el total.

Tabla 10: Cuenta total de puntos de función.

	Baja	Media	Alta	Total
EI	EI * 3	EI * 4	EI * 6	Σ EI
EO	EO * 4	EO * 5	EO * 7	Σ EO
EQ	EQ * 3	EQ * 4	EQ * 6	Σ EQ
ILF	ILF * 7	ILF * 10	ILF * 15	Σ ILF
EIF	EIF * 5	EIF * 7	EIF * 10	Σ EIF
Total (PESA)				Σ PFSA

6. Establecer el valor del factor de ajuste.

   Se obtiene el GTI (Grado Total de Influencia) mediante una ponderación de 0 a 5 de 14 factores que completan la visión externa de la aplicación.

Tabla 11: Factores de complejidad.

#	Factor de Complejidad	Valor
1	Comunicación de datos	
2	Proceso distributivo	
3	Objetivos de rendimiento	
4	Configuración de explotación usada por otros sistemas.	
5	Tasa de transacciones	
6	Entrada de datos en línea	
7	Eficiencia con el usuario final	
8	Actualizaciones en línea	
9	Lógica del Proceso Interno Compleja	
10	Reusabilidad del código	
11	Contempla la conversión e instalación	
12	Facilidad de operación	
13	Instalaciones múltiples	
14	Facilidad de cambios	
Factor de Complejidad Total (GTI)		Σ valor de los factores

7. Establecer los puntos de función ajustados.

   Luego de ser evaluadas las 14 características detalladas en el punto anterior se suman para obtener el GTI. Consecutivamente se obtiene el FAV (Factor de Ajuste de Valor) mediante el uso del GTI que se aplica en la siguiente ecuación.

   FAV = (GTI * 0,001) + 0,65

   Del total de puntos de función ajustados se utiliza la siguiente formula.

   PF = FAV * PFSA

   El cálculo del esfuerzo en horas – hombre se calcula mediante la siguiente ecuación:

$$Esfuerzo = \frac{PF}{\frac{1}{\#personas}} horas - \text{hom}bres$$

   Para calcular la duración en horas o en meses se hace uso de las siguientes ecuaciones.

$$\textbf{Duración en horas} = \frac{esfuerzo}{0{,}125}$$

$$\textbf{Duración en meses} = \frac{Duración_{horas}}{100 horas / mes}$$

Costo total = sueldo de 1 participante * #personas * duración en meses + otros costos

Las ventajas que se puede obtener con este modelo son la estimación de los puntos de función en el ciclo de vida durante el tiempo de definición de requerimientos, análisis y diseño, no se relacionan con el uso del lenguaje, tecnologías o herramientas, facilitan al personal no técnico una mejor comprensión de lo que se está midiendo ya que se basan en la vista de un usuario externo al sistema. Respecto a las desventajas sobresale el conteo subjetivo, tratándose del mismo problema se puede dar el caso que distintas personas obtengan diferentes estimaciones, es complicado de automatizar y de calcular, y es orientado a las aplicaciones de procesamiento de datos tradicionales.

4.1.3.3. Modelo basado en puntos de casos de uso

La estimación de costos a través del método basado en puntos de casos de uso, radica en medir el tiempo de desarrollo de un proyecto mediante el proceso de asignación de pesos a un determinado número de factores que lo afectan.

En concreto, este método adquiere como entrada los requisitos del sistema como actores y casos de uso, dando como resultado uno o más escenarios que determinan la manera que debe interactuar el sistema en relación al usuario o con otro sistema para lograr un objetivo específico.

En un caso de uso se registra una interacción entre el software y uno o más actores, esta interacción en un inicio se trata de una función autónoma dentro del sistema que facilita la estimación del tamaño del software en horas necesarias para la operación de los casos de uso y el número de personas que se necesitan para elaborarlo, midiendo la complejidad del sistema.

Las ventajas que se destacan de este modelo es que permite trabajar con diferentes tipos de software, presenta un buen rendimiento independientemente del tamaño del proyecto. Por otro lado, la desventaja que se presenta es que a pesar de la existencia del estándar UML para escribir casos de uso, cada ingeniero de acuerdo a su comprensión de los requerimientos del sistema escribe el caso de uso. A continuación, se describe el proceso a seguir:

1. Cálculo de los puntos de casos de uso no ajustados (UUCP, Unajusted Use Case Points).

Se trata de calcular el peso de los actores (UAW, Unajusted Actor Weights) y de los casos de uso (UUCW, Unajusted Use Case Weights) y posteriormente sumar el resultado de UAW y UUCW. Los criterios para establecer los pesos de los actores se muestran en la siguiente tabla.

Tabla 12: Clasificación y peso de los actores.

Tipo de actor	Descripción	Peso
Simple	Otro sistema externo, interactúa con el sistema a desarrollar mediante una interfaz de programación definida y conocida, (API, Application Programming Interface)	1
Medio	Otro sistema externo que interactúa a través de protocolo (conjunto de reglas que especifican el intercambio de datos u órdenes durante la comunicación entre las entidades que forman parte de la red)	2
Complejo	Un usuario físico que interactúa a través de una interfaz gráfica de usuario.	3

Se contabiliza el número de actores que hay en el sistema, para multiplicar cada tipo por su factor de peso y por último sumar esos productos para tener como resultado el UAW, como se presenta en la siguiente ecuación.

$$UUCW = \sum_{i=1}^{n=actores} (cantidadtipoactor_i * peso)$$

Por otra parte, los criterios para establecer los pesos de los actores se muestran en la siguiente tabla.

Tabla 13: Clasificación y peso de los casos de uso.

Tipo de caso de uso	Descripción	Peso
Simple	El caso de uso contiene menos de 3 transacciones	5
Medio	El caso de uso contiene de 4 a 7 transacciones	10
Complejo	El caso de uso tiene más de 7 transacciones	15

Al igual que los actores, es necesario contabilizar el número de casos de uso que posee el sistema, multiplicar cada tipo por su factor de peso y por último sumar esos productos para tener como resultado el UUCW.

$$UUCW = \sum \left(tipocasodeuso_i * peso \right)$$

Finalmente, se aplica la fórmula de puntos de casos de uso no ajustados UUCP.

$$UUCP = AUW + UUCW$$

2. Calcular los puntos de casos de uso (UCP, Use Case Points).

   Se trata de elaborar el producto de los puntos de casos de uso no ajustados, el peso de los factores técnicos (TCF, Technical Factors) presentadosenlatabladefactorestécnicos,http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2227-18992016000100018%20-%20t10 y el peso de los factores ambientales (EF, Enviroment Factors) presentados en la tabla de factores ambientales, los cuales se ponderan de acuerdo a las habilidades y experiencias del grupo o equipo de trabajo. Para calcular los TCF, se debe tomar en cuenta los valores entre 0 y 5, donde: Irrelevante de 0 a 2, Medio de 3 a 4, Esencial 5.

Tabla 14: Factores técnicos.

Factor	Descripción	Peso
T1	Sistema distribuido	2
T2	Objetivos de perfomance o tiempo de respuesta	1
T3	Eficiencia del usuario final	1
T4	Procesamiento interno complejo	1
T5	El código debe ser reutiliz-able	1
T6	Facilidad de instalación	0,5
T7	Facilidad de uso	0,5
T8	Portabilidad	2
T9	Facilidad de cambio	1
T10	Concurrencia	1
T11	Objetivos especiales de seguridad	1
T12	Acceso directo a terceras partes	1
T13	Facilidades especiales de entrenamiento a usuarios	1

Tabla 15: Factores ambientales.

Factor	Descripción	Peso
E1	Familiaridad con el modelo del proyecto utilizado	1,5
E2	Experiencia en la aplicación	0,5
E3	Experiencia en orientación de objetos	1
E4	Capacidad del análisis líder	0,5
E5	Motivación	1
E6	Estabilidad en los requer-imientos	2
E7	Personal de medio tiempo	-1
E8	Dificultad en el lenguaje de programación	-1

Consecutivamente, se procede a realizar la multiplicación del peso de cada factor por el respectivo nivel establecido por los valores de la tabla de factores técnicos, se suma esos productos para tener como resultado el TFactor. La fórmula equivalente se presenta a continuación:

$$TFactor = \sum \left(peso * nivel \right)$$

Finalmente, se aplica la siguiente fórmula de peso de factores técnicos (TCF).

$$TCF = 0{,}6 + \left(0{,}01 * TFactor \right)$$

De la misma manera que los TCF se debe tomar en cuenta los valores para estimar cada factor entre 0 y 5.

A continuación, se efectúa la multiplicación del peso de cada factor por el respectivo nivel establecido en la tabla de salario por lenguaje de programación.

Lenguaje	Muestra	Mediana	Media	Desviación estándar
Objetive C	21	$33.000	$40.141	$29.611
C	52	$25.725	$24.989	$13.762
Java	321	$25.000	$26.227	$16.342
VB	103	$25.000	$23.922	$11.809
C#	293	$24.000	$24.637	$13.549
Node JS	30	$23.970	$28.952	$20.037
PL/SQL	226	$23.450	$25.379	$16.436
Ruby	58	$23.250	$26.796	$19.870
JavaScript	429	$22.700	$24.463	$14.372
Bash	47	$21.500	$24.959	$11.466
Python	40	$20.000	$23.910	$19.481
PHP	167	$18.000	$20.074	$14.016
Delphi	18	$18.000	$20.028	$9.681

Se suman los productos para tener como resultado el EFactor. La fórmula equivalente se presenta a continuación:

$$EFactor = \sum \left(peso * nivel \right)$$

Luego, se aplica la fórmula de peso de factores técnicos (TCF).

$$EF = 1{,}4 + \left(-0{,}03 * EFactor \right)$$

Finalmente, para terminar con el paso 2, se aplica la fórmula de puntos de casos de uso (UCP).

$$UCP = UUCP * TCF * EF$$

3. Luego de obtener el valor de los puntos de casos de uso (UCP), calcula las horas-hombre aplicando la siguiente fórmula.

$$Horas - Hombre = UCP * 20$$

El autor de la técnica propone usar 20 horas / hombre por UCP (Use Case Points). Un ejemplo es que para un sistema de 60 UCP*20 horas / hombre se obtiene un total de 1200 horas / hombre. Lo cual corresponde a 30 semanas, de esta manera, un equipo de 5 personas desarrollaría el sistema en 6 semanas. En otras palabras, 75 semanas a 40 horas por semana para una única persona o 15 semanas para un equipo de 5 personas de tiempo completo.

Los tiempos estimados por etapa ya sea de análisis, diseño, codificación, etc., así como los costos por hora son criterio del equipo de desarrollo y dependen de manera muy significativa de su experiencia.

4.1.4. Velocidad del proyecto

La velocidad es una medida de la capacidad del equipo para terminar un trabajo en una determinada iteración.

La planificación se puede realizar en base en el tiempo o el alcance. La velocidad del proyecto se usa para establecer cuántas historias es posible implementar antes de una fecha planteada o cuánto tiempo llevará en implementar un conjunto de historias. En el caso de planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Por otro lado, al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario entre la velocidad del proyecto, obteniendo como resultado el número de iteraciones para su implementación.

La velocidad se formula a través de un rango, por ejemplo, de 20 a 30 puntos de historia por iteración, sobre todo al inicio de un proyecto si el equipo no ha resuelto un problema similar, es difícil determinar cuál sería la velocidad del equipo. Cabe recalcar que se refiere a la velocidad de un equipo y no la de un individuo.

Al inicio, existe la obligación de definir un rango de velocidad con la existencia de pocos datos. Se puede tomar en cuenta valores históricos, en caso de que el equipo y tamaño del problema sean iguales, lo cual es poco probable que exista. Por otro lado, se puede elaborar una iteración para obtener una idea de la velocidad con un equipo que esté trabajando en el proyecto, sin embargo, esto conlleva mayores costos y no funciona correctamente tratándose de tomar decisiones acerca de iniciar un proyecto.

Existe la posibilidad de realizar un pronóstico de la velocidad, esto incluye tomar varias iteraciones de historias y fragmentarlos en tareas que se realizan para completar la historia. Para cada tarea se estimaría el número de horas, lo cual incluye desarrollo, diseño, testing, etc., y se procedería a evaluar la capacidad que el equipo tendría en una iteración determinada. Se considera que el 70% es la base de capacidad para un buen equipo. Por ello, tomando en cuenta una situación regular, el total de horas disponibles de un equipo es, por ejemplo:

- 4 miembros de equipo x 2 semanas x 40 horas por semanas = 320 horas.
- Multiplicado por el 70% de capacidad = 224 horas.
- Sumado las tareas de características y deja de contar en 224.
- Con las características completadas, se suma los puntos de historia y se obtiene la velocidad, digamos 36.
- Designando el 20% a cada lado para obtener un rango de lo más

bajo y alto, para llegar a una velocidad estimada de 29 a 43 puntos de historia.

- La velocidad usualmente varía en las primeras 2 a 4 iteraciones y luego se estabiliza dentro de un pequeño rango de puntos. Por ello, se tiene un rango inicial antes de que el primer sprint fuera 29 a 43, al llegar al sprint 4, puede estabilizarse de 34 a 38. Lo que nos da más confianza en el pronóstico de la fecha de finalización.

4.1.5. Escenarios de Bechmarking

En esta etapa se busca estudiar a la competencia para mejorar los propios procesos, debido a esto, es necesario hacer una comparación y medición con respecto a otras empresas, teniendo en cuenta que cada empresa puede utilizar un tipo u otro de Benchmarking en función de sus intereses.

"La principal función de esta técnica consiste en propiciar el cambio a mejor por parte de la organización." [36]

Según [36] Existen varios tipos de Benchmarking que se puede llevar a cabo en una organización, en base a cuatro categorías diferentes:

- **Benchmarking interno:** Este proceso se lleva a cabo dentro de la propia organización. Se hace necesario realizarlo cuando en una organización los departamentos llevan consigo varias actividades o funciones similares en algunos aspectos. Ayuda a ver de manera más sencilla las mejores prácticas de la organización que serán adaptadas y transferidas a las áreas que puedan beneficiarse. Otros tipos pueden ser más eficientes y eficaces a diferencia de los de otras áreas de la misma empresa.
- **Benchmarking competitivo**: Logra dar a conocer productos, servicios y procesos de la competencia directa con la empresa y hace una comparación con los propios. Aunque no todos los secretos de las compañías rivales son fáciles de identificar, la respuesta a muchos interrogantes sobre nuestros competidores la tienen los propios consumidores. La aplicación de estudios de mercado diseñados para conocer la percepción que tienen los clientes sobre nuestra compañía en función de otras es una forma de iniciar este análisis con la competencia.
- **Benchmarking funcional:** Establece la identificación de productos, servicios y procesos de trabajo que poseen las empresas, aunque no necesariamente debe ser la competencia directa. Varios de sus funciones o procesos en los negocios son los mismos, pero se suelen tener independencia en las disimilitudes de las industrias.

Este modelo de Benchmarking resulta beneficioso, debido a que se pueden descubrir prácticas y métodos que no se implementan en la industria propia del investigador.

- **Benchmarking Word Class:** Es la más ambiciosa. Consiste en analizar a la organización que se desenvuelve mejor escala global, lo que supone un gran nivel de análisis. Este proceso requiere tener en cuenta los competidores referentes, empezando por un análisis exhaustivo del proceso, con el objetivo de aprender y obtener información que ayude a la organización a desarrollar acciones de mejora. Su éxito dependerá de la capacidad de la organización para gestionar la información de manera eficaz.

4.1.5.1. ¿Cómo se realiza?

- Identificar qué proceso, área o producto se desea mejorar en la organización.
- Identificar las empresas que posean las mejores prácticas y ver si efectivamente la comparación pudiera servir.
- Definir qué indicadores se va a medir.
- Definir el método para recopilar datos.
- Analizar la discrepancia de lo medido con el desempeño actual propio.
- Proponer los niveles de desempeño futuro de acuerdo a la comparativa, a las posibilidades y a los agregados de valor.
- Fijar las metas y tiempos de integración de la nueva práctica.
- Ejecutar la integración.
- Medir los resultados.
- Fijar la periodicidad con la que se realizará el análisis de mejores prácticas, debido que es un proceso de mejora continua.

Cabe resaltar que antes de inmiscuirse en un benchmark, hay que estar seguros de haber buscado todas las fuentes de información que se puedan obtener a través de los datos con los cuales se puede hacer cambios. Los centros de estadística, la academia, las OSC, algunas empresas y otras organizaciones publican documentos que pudieran servir a este fin. [37]

Ejemplo de historias de usuario

A continuación, se presenta dos ejemplos de historias de usuario:

Tabla 16: Ejemplo 1 de Historias de Usuario.

Historia de Usuario	
Número: 1	***Usuario:*** Cliente
Nombre de la historia: Cambiar dirección de envío	
Prioridad en negocio: Alta	***Riesgo en desarrollo:*** Baja
Puntos estimados: 2	***Iteración asignada:*** 1
Programador responsable: Juan Pérez	
Descripción: El cliente puede cambiar la dirección de entrega de cualquiera de los pedidos de envío.	
Observaciones:	

Tabla 17: Ejemplo 2 de Historias de Usuario.

Historia de Usuario	
Número: 2	***Usuario:*** Administrador, Usuario
Nombre de la historia: Autentificar Usuario	
Prioridad en negocio: Alta	***Riesgo en desarrollo:*** Baja
Puntos estimados: 2	***Iteración asignada:*** 1
Programador responsable: Juan Pérez	
Descripción: Permite acceder a la aplicación una vez registrado correctamente y poder asignar los permisos de acuerdo el rol ya sea administrador o usuario.	
Observaciones:	

Ejemplo del Análisis de benchmarking

Tomando como ejemplo un benchmarking para sector tecnológico [49], en este caso Apple, Samsung y Microsoft, debido a sus prestaciones, calidad, y estrategias de marketing.

En primer lugar, podemos comparar los scores de cada una de las webs de las empresas escogidas:

<table>
<tr><th>Apple</th><th>Samsung</th></tr>
<tr><td>Como podemos ver, el score de la web de líder es de un 60.8, un poco bajo, ya que cuenta con algunos errores y aspectos a mejorar.
  </td><td>Por otro lado, Samsung tiene un score de 81, es decir tiene una web bastante mejor que Apple, pero aun así tiene algunos aspectos para corregir.
  </td></tr>
<tr><th colspan="2">Microsoft</th></tr>
<tr><td colspan="2">Por último, la empresa Microsoft cuenta con un 78.5 de score, superando también a Apple y muy cerca de Samsung.
  </td></tr>
</table>

Se define el indicador a medir, para la realización de este ejemplo el cual es “Usabilidad”, cabe destacar que, para un buen análisis de benchmarking, se debe analizar todos los indicadores que sea posible:

Tabla 18: Ejemplo de benchmarking en “usabilidad”.

	Apple	Samsung	Microsoft
URL			
Favicon			
Página 404 personalizada			
Tamaño de página			
Tiempo de carga			
Idioma			
Marcado de datos Estructurado			
Indicador de fiabilidad			

Conclusión:

En este capítulo se proporciona un marco de trabajo que permite al gestor hacer estimaciones razonables de recursos, entregables, costos y planificación temporal. Estas estimaciones se hacen dentro de un marco de tiempo limitado al iniciar un proyecto de software, y tendrán que ser actualizadas constantemente a medida que avanza el proyecto. También, las estimaciones deberían definir los escenarios del mejor caso, y peor caso, de modo que los resultados del proyecto pueden limitarse.

CAPÍTULO 5: FASE DE DISEÑO

5.1. Fase de diseño

Para el diseño en la metodología SNAIL se toma en cuenta solo aquellas historias de usuario que el cliente ha escogido para la iteración propuesta, esto por dos motivos:

1. Se considera que desde un principio es poco probable disponer de un diseño completo del sistema y que se encuentre libre de errores.
2. Tomando en cuenta la naturaleza cambiante del proyecto, el realizar un diseño muy completo o extenso en las fases iniciales del proyecto para posteriormente corregirlo o cambiarlo, se considera una pérdida de tiempo

Es necesario destacar que esta actividad es constante en la duración del tiempo de vida del proyecto arrancando de un diseño inicial que es corregido y mejorado mediante el transcurso del proyecto.

Esta metodología se enfoca en diseños simples y entendibles. Los conceptos más importantes de diseño en esta metodología son los siguientes:

- **Simplicidad en el diseño**

Un aspecto muy importante dentro de esta metodología es la simplicidad, se considera que un diseño simple se implementa en menor tiempo comparado con un diseño complejo. Por tanto, la idea primordial es implementar el diseño más simple posible que funcione correctamente y que cumpla con los requisitos de las historias de usuario. No se recomienda adelantar la implementación de funcionalidades que no se han tomado en cuenta en la iteración en progreso. Respecto a los diagramas, se pueden utilizar considerando que no incluya mucho tiempo el realizarlos, deben garantizar una verdadera utilidad y que sean desechables, en esta metodología se considera muy importante disponer de una descripción clara del sistema o parte del sistema, en lugar de un conjunto de complejos diagramas que quizás tomen más inversión de tiempo y sean de menor utilidad.

- **Soluciones puntuales**

En muchos de los casos los equipos de desarrollo se ven desafiados por requerimientos de los clientes o historias de usuario los cuales presentan problemas desde la perspectiva del diseño o la implementación. Las soluciones puntuales, son pequeños programas de prueba que permiten solucionar este tipo de inconvenientes, son una pequeña aplicación totalmente desconectada del proyecto con la cual se pretende analizar el problema y luego se propone una solución viable al mismo. Puede ser rústica y sencilla, pero debe brindar la información necesaria para resolver el problema encontrado.

- **Recodificación o refactorización**

Como se había mencionado el diseño de software de la metodología SNAIL, se trata de una actividad constante en la duración del tiempo de vida del proyecto y la refactorización complementa este concepto. Al igual que cualquier metodología tradicional en SNAIL se comienza el proceso de desarrollo con un diseño inicial. La diferencia es que en las metodologías tradicionales se trata de un diseño muy complejo debido a que es completo y grande lógicamente se ha invertido mucho tiempo en su elaboración considerando el caso que si se da su modificación será un fracaso notable para el grupo de desarrollo. Por otro lado, el caso de la metodología SNAIL es todo lo opuesto a lo mencionado, tratándose de un diseño inicial muy general y sencillo incluye mucho tiempo para su elaboración, al mismo que se realizan adiciones y correcciones en el progreso del proyecto, con el objetivo de mantener una aplicación correcta y a la vez simple.

La refactorización se relaciona directamente con el código procurando que este sea sencillo lo cual permita mantenerlo de forma fácil. En cada revisión que se presente algún tipo de redundancia, funcionalidad innecesaria o cualquier tipo de problema a solucionarse, se debe corregir esa unidad de código con el objetivo de alcanzar un código sencillo en términos de lectura y mantenimiento. Es difícil aplicar estas prácticas cuando se está iniciando en SNAIL por situaciones como la desconfianza que existe en los equipos de desarrollo al tratarse de modificar algo existente que funciona bien tanto a nivel de diseño o implementación. Mientras tanto, en el caso de disponer de un panorama de pruebas completo y un sistema de automatización para las mismas es probable lograr éxito en el proceso. Otra situación es que se considera que es mucho más el tiempo que se pierde en refactorización que el obtenido en sencillez y mantenimiento. Según SNAIL la ventaja obtenida en refactorización es muy significativa que justifica adecuadamente el esfuerzo adicional en corrección de redundancias y funcionalidades innecesarias.

- **Metáforas del sistema**

La metodología SNAIL recomienda aplicar el concepto de metáforas como una forma sencilla de plasmar el propósito del proyecto, y su vez dirigir la estructura y arquitectura del mismo. Un ejemplo claro, sería realizar un registro de los nombres o nomenclatura de las clases o métodos que se implementen en el código. Para que existe una buena comunicación entre el cliente y el grupo de desarrolladores es necesario compartir esta metáfora, Una metáfora debe brindar una fácil comprensión para el cliente y al mismo tiempo dispone de información completa para la arquitectura del proyecto. Es importantes resaltar que al elaborar una metáfora es oportuno escribir nombres adecuados a todos los elementos del sistema permanentemente, y que estos se pertenezcan a un sistema de nombres adecuados. Esto permitirá determinar aspectos importantes en fases posteriores del desarrollo.

5.1.1. Diseño de bases de datos

El proceso del diseño de la base de datos está compuesto por subprocesos que inician a partir de la recopilación de requisitos de la base de datos. Esto radica en la elaboración de tres diseños de base de datos los cuales son: conceptual, lógico y físico los mismos que se realizan mediante el uso de técnicas y métodos específicos.

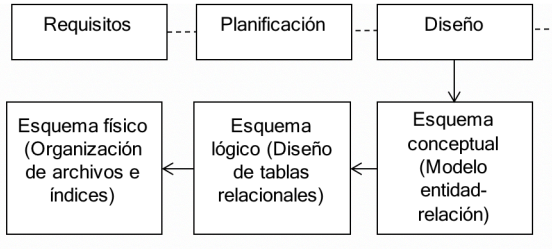

Ilustración 7: Proceso de desarrollo de la base de datos.

- **Diseño conceptual**

El objetivo del diseño conceptual es definir las entidades y relaciones de manera abstracta, sin regirse en algún modelo lógico en específico.

En este diseño se elabora un esquema de la información del entorno o el sistema donde se establecerá la base de datos, se pretende obtener la comprensión de la estructura, semántica, relaciones y restricciones de la base de datos, crear una descripción constante de la información de la base de datos, obtener la comunicación entre usuarios, analistas y diseñadores, para de esta manera continuar con el diseño lógico. Cabe recalcar que este diseño conceptual se desarrolla en base a la información de los requisitos de usuario.

- **Diseño lógico**

El diseño lógico es el proceso de crear un esquema de información, pero en este caso en base a un modelo de base de datos específico, particularmente del Sistema Gestor de Base de Datos que se manipulará o de cualquier otra consideración física. En este diseño, se lleva a cabo la transformación del esquema conceptual en un esquema lógico con las estructuras de datos del modelo de base de datos en el que se basa el Sistema Gestor de Base de Datos objetivo. Los resultados obtenidos son un grupo de estructuras propias del modelo abstracto de datos, como son las tablas en las bases de datos relacionales.

- **Diseño físico**

En el diseño físico se realiza un esquema físico, a partir del esquema lógico. Es en el proceso donde se lleva a cabo implementación de la base de datos, las estructuras de almacenamiento y los métodos para acceder a la información. Para iniciar con esta etapa, se debe haber elegido el Sistema Gestor de Base Datos, puesto que el esquema físico se adecúa a este. Existe una realimentación entre el diseño físico y el diseño lógico, debido a que las decisiones que se consideren durante el diseño físico para mejorar las prestaciones, pueden afectar a la estructura del esquema lógico. Al culminar estas fases el sistema de base de datos prácticamente está terminado, sin embargo, en la mayoría de los casos es necesario realizar cambios el diseño de la base de datos luego de su funcionamiento.

5.1.2. Diccionario de datos

El diccionario de datos se define como lista estructurada donde se describen todos datos que forman parte del sistema, con un conjunto de definiciones claras y rigurosas con el fin que el analista y el usuario entiendan las entradas, salidas, elementos de los almacenamientos y cálculos intermedios. Esto incluye almacenes de datos, flujos de datos, estructuras de datos, elementos de datos y en algunos casos el modelo E-R.

El diccionario de datos define los datos con el fin de:

1. Describir el significado de los flujos de datos.
2. Describir la estructura de datos que se presenta en transcurso de los flujos.
3. Describir la estructura de datos en los almacenes.
4. Describir los detalles de las relaciones entre almacenes que se presentan en un diagrama entidad-relación.

Los diccionarios de datos son utilizados por cuatro razones importantes:

1. Para manipular los detalles en sistemas extensos debido a que resulta complicado recordar todo lo referente a un sistema.
2. Para comunicar un significado habitual respecto a todos los componentes del sistema. Esto resulta importante cuando trabajan muchos desarrolladores y no se reúnen con frecuencia para comunicarse.
3. Para documentar las características del sistema.
4. Para localizar de manera más sencilla errores en el sistema.

5.1.2.1. Contenido de un Diccionario de Datos

El diccionario de datos contiene las siguientes:

1. Definiciones lógicas de datos:

- Elemento de Dato (Atributos de la Entidad).
- Estructura de Dato.
- Flujos de Datos.
- Almacenes de datos.

2. Definiciones lógicas de procesos.
3. Definición lógica de entidad externa.

5.1.2.2. Notación del Diccionario de datos

1. Descripción de los datos elementales:

- Nombre: para diferenciar los datos elementales se asigna nombres que sean significativos, en otras palabras, debe poseer un nombre único en el contexto del desarrollo del sistema. Por ejemplo: fecha factura. Un nombre no debe ser mayor de 30 caracteres y no debe tener espacios en blanco.
- Descripción: indica de manera breve y clara lo que el elemento representa dentro del sistema, además debe ser comprensible para el lector.
- Alias: los alias son opcionales, es cuando el mismo dato tiene varios nombres, según quien haga uso del dato. Por ejemplo; factura puede tener como alias documento de pago o nota de pago. No se considera alias los siguientes casos: factura autorizada, factura verificada.
- Longitud: indica el tamaño o espacio de caracteres que ocupa cada dato sin considerar la forma en que serán almacenados.
- Valores: si los valores de los datos están limitados a un intervalo específico, debe demostrarse en la entrada del diccionario de datos. Por ejemplo, Longitud unidad [centímetros], rango [1-100].

2. Descripción de las estructuras de datos: las estructuras de datos se edifican sobre cuatro relaciones de componentes los cuales son:

- Relación secuencial: define los datos o estructuras que siempre se incluyen en una estructura de datos en particular, en otras palabras, también se llama concatenación de dos o más datos.

- Relación de selección: define opciones para datos o estructuras comprendidas en una estructura de datos.
- Relación de iteración: define la duplicación de un componente de cero o más veces.
- Relación opcional: es un caso particular de la iteración, es decir, una o ninguna iteración.

3. **Descripción de los flujos de datos:** representa los flujos de datos mientras el flujo no sea un único atributo. Está constituido por una o más estructuras que han sido definidas.

- Nombre del flujo de datos: se asigna nombres representativos, es decir, que tengan significado único en el contexto del desarrollo del sistema. Por ejemplo: factura.
- Fuente: establece el proceso fuente de la información. Se establecerá el número del proceso.
- Destino: estable el proceso destino de la información. Se establecerá el número del proceso.
- Definición: describe el contenido del flujo de datos.
- Contenido: detalla las estructuras de datos incluidas.

4. **Descripción de los almacenamientos de datos:**

- Nombre de almacenamiento de datos: se define nombres que sean representativos, es decir, que sea un significado único dentro del desarrollo del sistema. Ejemplo: histórico facturas.
- Flujos de entrada: establece los flujos que mantienen el almacenamiento de datos.
- Flujos de salida: establece los flujos que extraen información del almacenamiento de datos.
- Definición: detalla el contenido del almacenamiento de datos.
- Contenido: explica el contenido del almacenamiento.

5. **Descripción de los procesos:** representa los procesos del sistema.

- Nombre de proceso: se establece nombres representativos, es decir, que sea que tengan un significado único dentro del desarrollo del sistema. Por ejemplo: verificar _crédito.
- Entradas: establece los procesos, almacenamientos de datos que trabajan como fuente de datos.

- Flujos de salida: establece los procesos, almacenamientos de datos que trabajan como destino de datos.
- Definición: establece la misión del proceso.
- Descripción: detalla el proceso. Para esto se hace uso de: forma narrativa, arboles de decisión, tablas de decisión, lenguaje estructurado.

6. **Descripción de las entidades externas:** representamos las entidades externas del sistema.

- Nombre de entidad externa: se establecen nombres representativos, que identifiquen a la entidad. Por ejemplo: clientes.
- Flujos de datos asociados: establece los flujos (entrada / salida) asociados.
- Definición: establece quienes son la entidad.
- Volumen: Número de componentes de la entidad.

5.1.3. Diseño conceptual

Esta fase es tomada de la metodología OOHDM [38] para el desarrollo de nuestra metodología híbrida. En ella se elabora o diseña un esquema conceptual representado por los objetos del dominio, las relaciones y colaboraciones existentes establecidas entre ellos. En las aplicaciones hipermedia tradicionales, en los cuales sus componentes de hipermedia no han sido reestructurados durante la ejecución, sería posible utilizar un modelo de datos semántico estructural (como el modelo de entidades y relaciones). De esta manera, en los casos en que la información base pueda ser cambiada dinámicamente o se intenten ejecutar cálculos complejos, se necesitará hacer más productivo el comportamiento del modelo de objetos.

En la metodología SNAIL, el esquema conceptual está construido por clases, relaciones y subsistemas. Las clases son descritas como en los modelos orientados a objetos tradicionales. Aun así, los atributos tienen la ventaja de poder ser de múltiples tipos para representar perspectivas diferentes de las mismas entidades del mundo real.

Se hace uso similar a UML (Lenguaje de Modelado Unificado) y tarjetas de clases y relaciones similares a las tarjetas CRC (Clase Responsabilidad Colaboración). Este esquema de las clases se basa en un conjunto de clases conectadas por relaciones. Los objetos son instancias de las clases. Las clases son usadas durante el diseño navegacional para derivar nodos, y las relaciones que son usadas para construir enlaces.

5.1.3.1. Implementación de capa conceptual

En este apartado, para su desarrollo se inicia diseñando la capa conceptual, Estableciendo que el principal objetivo de esta etapa es capturar los conceptos involucrados en el dominio de la aplicación y describirlos con un mayor detalle, utilizando a los diagramas que permitan expresar con claridad el comportamiento, la estructura y las relaciones entre dichos conceptos. La Programación Orientada a Objetos hace fácil el traspaso del diseño conceptual a la implementación, dándole al programador herramientas que permiten reducir la distancia entre el problema del mundo real y la programación de la solución en la computadora.

En cada diseño conceptual hay comportamiento que escapa a la simple navegación de información. Este se trata del comportamiento inherente de cada clase, y aunque la aplicación particular no necesita que se implemente, vale destacar que si la aplicación crece el diseño conceptual debe estar preparado para ser extendido, tal como cualquier diseño orientado a objetos.

5.1.4. Diseño Navegacional

Un modelo navegacional se construye a manera de una vista sobre un diseño conceptual, permitiendo la construcción de modelos diferentes de acuerdo con los diferentes perfiles de usuarios. Cada modelo navegacional suministra una vista subjetiva del diseño conceptual.

El diseño de navegación se expresa en dos esquemas: el esquema de clases navegacionales y el esquema de contextos navegacionales. Se tiene un conjunto de tipos predefinidos de clases navegacionales: nodos, enlaces y estructuras de acceso. El significado de los nodos y los enlaces son los tradicionales de las aplicaciones hipermedia, y las estructuras de acceso, como lo son los índices o recorridos guiados, representan los posibles caminos de acceso a los nodos.

La estructura primitiva base del espacio navegacional es la noción de contexto navegacional. Un contexto navegacional es un conjunto de nodos, enlaces, clases de contextos, y otros contextos navegacionales (contextos anidados). Pueden ser definidos por comprensión o extensión, o por enumeración de sus miembros.

Los contextos navegacionales tienen un papel principal similar a las colecciones y están diseñadas en base el concepto de contextos anidados. Organizan el espacio navegacional en grupos convenientes que pueden ser recorridos en un orden particular y que deberían ser definidos como caminos para beneficiar al usuario mediante la ayuda logrando la tarea deseada.

Los nodos son enriquecidos con un conjunto de clases especiales que permiten de un nodo observar y presentar atributos (incluidos las anclas), así como métodos (comportamiento) cuando se navega en un particular contexto.

5.1.4.1. Implementación de capa Navegacional

La capa navegacional se compone de objetos construidos a partir de objetos conceptuales, y constituyen en general los elementos canónicos de las aplicaciones hipermedia tradicionales: nodos, enlaces, anclas y estructuras de acceso. Sin embargo, estas clases pueden extender el comportamiento característico para funcionar como adaptadores [39] de los objetos conceptuales y delegar así operaciones específicas del dominio.

Es por esto, que los objetos navegacionales pueden actuar como observadores, construyendo vistas de objetos conceptuales, y como adaptadores, para extender la actividad navegacional de un nodo y poder aprovechar el comportamiento conceptual del objeto adaptado.

5.1.5. Diseño de interfaz abstracta

Una vez definido el modelo navegacional, se deben especificar los aspectos de interfaz. Esto quiere decir que se debe definir la forma en la cual los objetos navegacionales pueden aparecer, ¿cómo los objetos de interfaz activarán la navegación y el resto de la funcionalidad de la aplicación?, ¿qué transformaciones de la interfaz son pertinentes y cuándo es necesario realizarlas?

Una clara diferencia entre diseño navegacional y diseño de interfaz abstracta hace posible construir diferentes interfaces para el mismo modelo navegacional, dejando un alto grado de independencia de la tecnología de interfaz de usuario.

La forma en cómo se visualiza la interfaz de usuario en aplicaciones interactivas (en particular las aplicaciones web) es muy importante en el desarrollo que las metodologías actuales tienden a descuidar. En esta metodología hibrida se utiliza el diseño de interfaz abstracta para describir la interfaz del usuario de la aplicación de hipermedia.

5.1.5.1. Implementación de capa abstracta

De la misma manera que un nodo actúa como observador, un nodo también actúa como adaptador, por último, continúa mostrando cierta información y para ello necesita definir la manera en cómo se presenta mediante la dicha información que será visualizada en la interfaz. Es por esto que se incorporan las dos últimas tecnologías a las que se hará mención en este apartado: XSL y un mecanismo de análisis sintáctico para obtener una página HTML en función de un par de documentos XML/XSL.

Las páginas XSL, ubicadas también del lado del servidor, definirán la apariencia de los nodos que se generaron en formato XML. Cada página XSL define la forma en que los elementos del XML asociados serán mostrados, haciendo uso de código HTML y

eventualmente CSS [40], para establecer el formato deseado a las últimas páginas. Los enlaces y estructuras de acceso también se van mostrando al igual con los estilos adoptados para este tipo de información navegacional.

La relación entre un XML y su apariencia puede establecerse en forma explícita, con una etiqueta detallada la cual esté ubicada en la primera línea del archivo XML, o bien podría ser calculado en forma dinámica para satisfacer demandas de personalización. Dado el tipo de caso, regresarle al cliente un archivo XML que haga referencia a un archivo XSL ubicado en el servidor tiene al menos dos inconvenientes: genera un tráfico de ida y vuelta innecesario, ya que podría evitarse enviando directamente el código HTML al cliente, y puede requerir características especiales en el explorador del cliente que pueden privarlo de visualizar la página correctamente.

Ejemplo del modelo conceptual:

El lenguaje elegido para desarrollar la implementación de esta fase de es Java. Cabe destacar que, durante esta etapa se utilizará también JDBC como paquete de vital importancia para el manejo de base de datos.

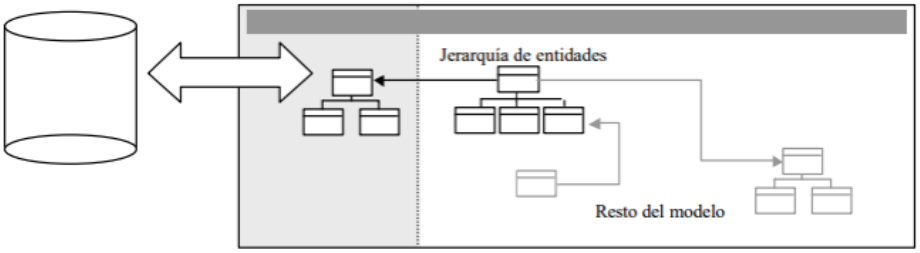

Ilustración 8: Paquete de interfaz con la base de datos, dentro del Diseño Conceptual.
Fuente: [38]

Cabe mencionar que el diseño conceptual puede estar compuesto de otras clases que por su forma no se pueden considerar subclases de Entidad/Abstracta. Estos casos son simplemente colaboradores de entidades concretas, clases que son necesarias para realizar algún tipo de actividad en específico y de una vida útil corta, o algunos casos de entidades débiles desde el punto de vista de diseño entidad-relación.

Tómese como ejemplo la información relacionada con la navegación del usuario en una determinada sesión, no es importante para otras sesiones hasta incluso para el resto de la aplicación. En estos supuestos, se requiere una clase para contener la información mencionada y el comportamiento para manipularla, pero no requiere considerar la persistencia dentro de su funcionalidad.

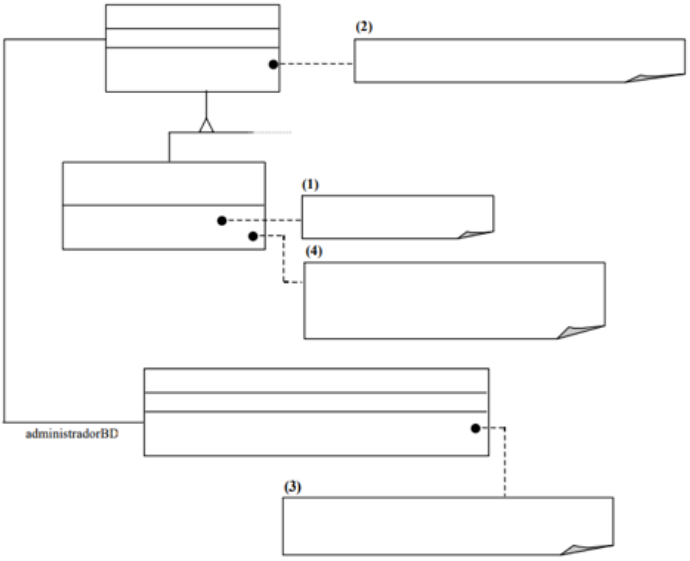

Ilustración 9: Instanciación de una subclase concreta de EntidadAbstracta.

Ejemplo del modelo Navegacional:

La ilustración 5 ilustra la construcción del nodo a partir de un requerimiento HTTP proveniente de un cliente remoto.

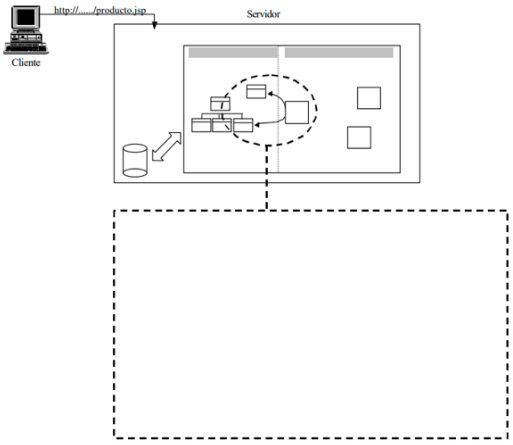

Ilustración 10: Construcción de un nodo de la capa navegacional.
Fuente: [38]

Como se puede observar, la cadena se inicia con un requerimiento del usuario que navega por la aplicación. Al momento de ingresar a un enlace, una página JSP sele a

flote para elaborar una página XML. En principio, el archivo XML generado puede ser útil para transferir información entre servidores cooperativos, o solo entre un cliente y un servidor mediante un esquema tradicional. En cualquier caso, las portabilidades brindadas por las características simples de XML hacen posible una transferencia transparente incluso entre plataformas completamente heterogéneas.

Ejemplo del modelo de interfaz abstracta:

La transformación del lado del servidor es provista en la actualidad por paquetes como Xalan-Java [51] y Saxon [52]. Un esquema de la operación de transformación puede observarse en la Figura 6.

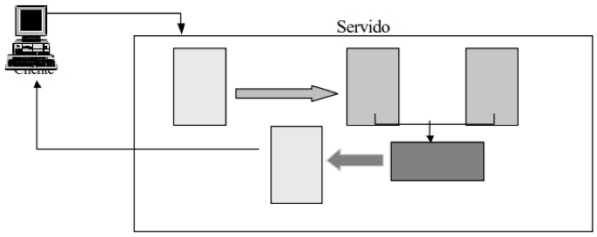

Ilustración 11: Generación de un documento HTML a partir de una fuente XML + XSL.
Fuente: [38]

El lenguaje XSLT (Transformaciones XSL) [53] es utilizado para componer hojas de estilo XSL. Estos documentos contienen instrucciones que mediante el uso de Xalan-Java, por ejemplo, sirven para llevar a cabo las transformaciones y producir un documento de salida, una secuencia de caracteres o de bytes, un DOM (Modelo de Objetos de Documento), etc. Dado el ejemplo que se ha mencionado anteriormente, se necesitaba obtener un documento de salida con formato HTML a partir de un par de documentos XML / XSL.

En esta fase concluye el capítulo 5, finalmente, las tres capas de diseño deberán ser implementadas por separado. Se ha intentado mantener la independencia de los elementos del dominio con las construcciones navegacionales y de presentación, haciendo uso de las tecnologías adecuadas para trabajar con la metodología propuesta, y aprovechar al máximo las virtudes de una aplicación de objetos.

Conclusión:

Este capítulo el diseño del sistema se reestructura y organiza en elementos que puedan ser desarrollados individualmente, aprovechando las ventajas del desarrollo en equipo. Es conveniente distinguir entre diseño de alto nivel o arquitectónico y diseño detallado. El primero, tiene como objetivo establecer la estructura de la solución (una vez que la fase de análisis haya definido concretamente el problema) identificando grandes módulos y sus relaciones. Con esto se estructura la arquitectura de la solución elegida. El segundo define los algoritmos empleados y la organización del código para comenzar la implementación.

CAPÍTULO 6: FASE DE PROGRAMACIÓN

6.1. Fase de programación

Esta es la fase en donde se transcribe el código fuente, haciendo uso de prototipos, así como de pruebas y ensayos para evitar tener errores y de tenerlos, lograr corregirlos a tiempo. Independientemente del lenguaje de programación que se maneje y su versión; se crean las bibliotecas y componentes que se reutilizarán dentro del mismo proyecto para hacer proceso mucho más rápido de la programación.

6.1.1. Codificación

Sin lugar a dudas, el cliente es una parte más del equipo de desarrollo; su presencia es indispensable en las distintas fases de SNAIL. Y al momento de codificar una historia de usuario su presencia es aún más necesaria. No se debe de olvidar que los clientes son los que crean las historias de usuario y acuerdan los tiempos en los que serán implementadas. Antes de empezar con el desarrollo de cada historia de usuario el cliente debe especificar de manera detallada lo que ésta hará y también deberá de estar presente cuando se realicen las pruebas (test) que validen que la historia que fue implementada cumpla con la funcionalidad especificada.

La codificación se debe de realizar ateniendo a estándares de codificación ya creados. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad.

Crear los test que prueben el funcionamiento de los distintos códigos implementados servirán de gran ayuda para desarrollar dicho código. Crearlos con previo aviso nos ayuda a saber qué es exactamente lo que tiene que hacer el código que se va a implementar y sabremos que una vez implementado pasará dichos test sin problemas ya que dicho código ha sido diseñado para ese fin. Se puede separar a la funcionalidad que debe cumplir una tarea a programar en pequeñas unidades, así, de esta manera se crearán primero los test para cada unidad y a continuación se desarrollará dicha unidad, así poco a poco se tendrá un desarrollo que cumpla todos los requisitos especificados.

Como ya se comentó anteriormente, S.N.A.I.L se decide por la programación en pareja (tomando en cuenta la fase que posee la metodología XP) ya que permite un código más eficiente y con una gran calidad.

SNAIL sugiere un modelo de trabajo el cual use repositorios de código dónde los programadores que se encuentran en parejas, puedan publicar a cada rango de tiempo sus códigos implementados y corregidos junto a los test que deben pasar. Así, de esta manera, el resto de programadores que necesiten códigos ajenos podrán trabajar siempre con las últimas versiones. Esto se realiza para mantener un código

consistente, poseer un código en un repositorio y publicarlo es una acción exclusiva para cada pareja de programadores.

SNAIL también propone un modelo de desarrollo en grupo, para que, en él, todos los programadores están inmersos en todas las tareas; cualquiera puede modificar o ampliar una clase o método de otro programador si es necesario y subirla al repositorio de código. Tener la opción de permitir al resto de los programadores modificar códigos que no son suyos no supone ningún tipo de riesgo debido a que, para que un código pueda ser publicado en el repositorio tiene que pasar los test de funcionamiento definidos para el mismo.

La mejora y optimización del código siempre tiene que dejar para el final debido a que, antes, hay que hacer que funcione y que sea correcto, pasando esto, se puede optimizar.

SNAIL afirma que la mayoría de los proyectos que necesiten de más tiempo del cual han sido planificado para ser finalizados no podrán ser terminados a tiempo se haga lo que se haga, así se contraten o ingresen más desarrolladores y se incrementen los recursos. La forma de solucionar este problema, lo plantea X.P y la cual se ha tomado muy en cuenta para esta metodología híbrida, y es realizar un nuevo "Release plan" para concretar los nuevos tiempos de publicación y de velocidad del proyecto.

Al momento de realizar el codifico no se sigue la regla de que propone la metodología X.P debido a que esta aconseja crear test de funcionamiento con entornos de desarrollo antes de programar. En SNAIL, nuestros test los obtendremos de la especificación de requisitos ya que en esta parte se especifican las pruebas que deben pasar las distintas funcionalidades del programa, procurando codificar pensando en las pruebas que debe pasar cada funcionalidad.

6.1.1.1. Recodificación

La recodificación ("refactoring") se base en traspasar o escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de hacerlo más simple, conciso y/o entendible. En varias ocasiones, al terminar de escribir un código de programa, se piensa que, si lo comenzáramos de nuevo, lo hubiéramos hecho en forma diferente, más clara y eficientemente. Pero, como ya está pronto y "funciona", rara vez es rescrito.

Las metodologías SNAIL sugieren recodificar cada vez que sea necesario. Si bien, parece una total pérdida de tiempo innecesaria en el plazo inmediato, el resultado que se obtiene de ésta práctica tienen sus recompensas, en las futuras iteraciones, cuando sea necesario ampliar o cambiar la funcionalidad. La idea que se persigue es, como ya se mencionó, tratar de mantener el código más simple posible que implemente la funcionalidad deseada.

6.1.1.2. Programación en pares

SNAIL "propone que se desarrolle en pares de programadores, ambos trabajando juntos en un mismo ordenador." [41] Aunque parezca que ésta práctica maximiza al doble el tiempo asignado al proyecto (y, por ende, los costos en recursos humanos), al trabajar en pares se minimizan los errores y se logran mejores diseños, compensando la inversión en horas. El producto obtenido es por lo general de mejor calidad que cuando el desarrollo se realiza por programadores individuales.

La programación en pares tiene un sobre costo aproximado de 15%, y no de un 100% como se puede pensar a priori. Este sobre costo es rápidamente pagado por la mejor calidad obtenida en el producto final.

Como un plus, la programación en pares tiene las siguientes ventajas:

- Gran parte de los errores son descubiertos en el momento en que se codifican, debido a que el código es permanentemente revisado por dos personas.
- El número de defectos hallados en las pruebas es estadísticamente menor.
- Los diseños son mejores y el código más corto.
- El equipo resuelve problemas en forma más rápida.
- Las personas aprenden significativamente más, acerca del sistema y acerca de desarrollo de software.
- El proyecto finaliza con más miembros que poseen conocimiento de los detalles de cada parte del código.
- Las personas aprenden a trabajar juntas, generando mejor dinámica de grupo y haciendo que la información fluya rápidamente.
- Las personas disfrutan más de su trabajo.

6.1.1.3. Disponibilidad del cliente

Uno de los requerimientos de SNAIL es tener al cliente disponible durante todo el proyecto. No solamente como apoyo a los desarrolladores, sino formando parte del grupo. El involucramiento del cliente es fundamental para que pueda desarrollarse un proyecto con esta metodología.

Al empezar con el proyecto, el cliente debe proporcionar las historias de usuarios.

Pero, dado que estas historias son expresamente cortas y de "alto nivel", no poseen los detalles necesarios para desarrollar del código. Estos detalles deben ser

proporcionados por el cliente, y discutidos con los desarrolladores, durante la etapa de desarrollo. No es necesario de largos documentos de especificaciones, solo que los detalles son proporcionados por el cliente, en el momento adecuado, "cara a cara" a los desarrolladores.

Explicado esto, esta fase parece demandar del cliente recursos por un tiempo prolongado, debe tenerse en cuenta que en otras metodologías este tiempo es insumido por el cliente en realizar los documentos detallados de especificación.

Adicionalmente, al estar el cliente en todo el proceso, puede prevenir a tiempo de situaciones no deseables, o de funcionamientos que no eran los que en realidad se deseaban. En otras metodologías, estas situaciones son detectadas en forma muy tardía del ciclo de desarrollo, y su corrección puede llegar a ser muy complicada.

6.1.2. Estándares

Si bien es conocido, esto no es una nueva propuesta, Pero SNAIL promueve la programación basada en estándares, Así, de esta forma, procura que sea fácilmente entendible por todo el equipo, y que facilite la recodificación.

6.1.3. Prueba Unitaria

Las pruebas unitarias es una de las bases fundamentales en SNAIL. Dado que todos los módulos deben de pasar las pruebas unitarias antes de ser liberados o publicados. Por otro lado, como se mencionó anteriormente, las pruebas deben ser definidas antes de realizar el código ("Test-driven Programming"). Esto quiere decir que todo código liberado va a pasar correctamente las pruebas unitarias, eso es lo que valida que funcione la propiedad colectiva del código. En este aspecto, el sistema y el conjunto de pruebas debe ser guardado en conjunto con el código, para que este pueda ser utilizado por otros desarrolladores, en caso de tener que corregir, cambiar o recodificar parte del mismo.

6.1.3.1. Programación dirigida por las pruebas

En las metodologías tradicionales, la fase de pruebas, incluyendo la definición de los test se realiza comúnmente casi el final del proyecto, o sobre el final del desarrollo de cada módulo. La metodología SNAIL propone un modelo ambiguo, en el que, lo primero que se escribe son los test que el sistema debe pasar. Luego, el desarrollo debe ser el mínimo necesario para pasar las pruebas previamente definidas.

Las pruebas de las que se está tratando en este libro, son las pruebas unitarias, realizados por los desarrolladores. La definición de estos test al comienzo, condiciona o "dirige" el desarrollo.

6.1.3.2. Detección y corrección de errores

Al momento que en esta fase se encuentra un error (“bug”), éste debe ser corregido inmediatamente, y se deben tener precauciones para que errores similares no vuelvan a ocurrir. De la misma manera, se generan nuevas pruebas para verificar que el error haya sido resuelto.

6.1.4. Interconexión

También conocida como “Estructura modular” de un proyecto o sistema; constituye la arquitectura de un programa orientado a la web y se puede representar de forma gráfica mediante un diagrama de estructura basado en la jerarquía, descomposición e independencia de los módulos.

En esta fase se representan los módulos, la interconexión entre los mismos y los parámetros de enlace entre ellos. Los módulos se representan por rectángulos con un nombre que indica la función del módulo. La interconexión entre módulos se representa por una línea que los conecta, siendo el que está por encima el que hace la llamada. Los parámetros se presentan por líneas que en un extremo terminan en punta de flecha para indicar la dirección del paso de variables y por el otro extremo terminan en un pequeño círculo. La estructura comienza con un módulo raíz o modulo principal, que es el primero que se ejecuta, y el que inicia la llamada a los otros.

6.1.5. Integración

6.1.5.1. Integración permanentes

Hay algo que se debe tener bien en claro, y es que todos los desarrolladores necesitan trabajar siempre con la “última versión”.

El realizar cambios o mejoras sobre versiones antiguas pueden llevar a causar graves problemas, y al retraso del proyecto. Es por eso que SNAIL incita a publicar lo antes posible las nuevas versiones, aunque no sean las últimas, siempre y cuando estén libres de errores. Idealmente, todos los días deben existir nuevas versiones publicadas.

Para no tener este o cualquier tipo de errores, solo una pareja de desarrolladores puede integrar su código a la vez.

Propiedad colectiva del código

En un proyecto, todo el equipo puede contribuir con nuevas ideas que apliquen a cualquier parte del proyecto. Asimismo, cualquier pareja de programadores puede cambiar el código que sea necesario para corregir problemas, agregar funciones o recodificar.

En otras metodologías, este concepto puede parecer extraño. Muchas veces se asume que, si hay algo de propiedad colectiva, la responsabilidad también es colectiva. Y que "todos sean responsables", muchas veces significa que "nadie es responsable". Este razonamiento no es correcto cuando se trabaja con la metodología de SNAIL. En este caso, quienes encuentran un problema, o necesitan desarrollar una nueva función, pueden resolverlo directamente, sin necesidad de "negociar" con el "dueño" o autor del módulo (ya que, de hecho, este concepto no existe en SNAIL).

Muchas veces, se explica que una solución pasa por la recodificación de varios módulos, que atraviesan de forma horizontal una determinada jerarquía vertical. Si es necesario dialogar y convencer al encargado de cada módulo, posiblemente la solución no se pueda implementar, por lo menos en tiempos razonables. En esta metodología, se promueve la recodificación, en aras de generar códigos más simples y adaptados a las realidades cambiantes. Cualquier pareja de programadores puede tomar la responsabilidad de este cambio. Los testeos permanentes deberían de aseguran que los cambios realizados cumplen con lo requerido, y, además, no afectan al resto de las funcionalidades.

Ritmo sostenido

La metodología especifica que debe llevarse un ritmo sostenido de trabajo.

En otras metodologías a ésta práctica se le denomina "Semana de 40 horas". Sin embargo, lo importante no es si se trabajan, 35, 40 o 42 horas por semana. El concepto que se desea establecer con esta práctica es el de planificar el trabajo de manera de mantener un ritmo constante y razonable, sin sobrecargar al equipo.

Al momento en que un proyecto se retrasa, trabajar tiempo extra puede ser más dañino que beneficioso. El trabajo extra desmotiva inmediatamente al grupo e impacta en la calidad del producto. En la medida de lo posible, se debería renegociar el plan de entregas ("Release Plan"), Desarrollando una nueva reunión de planificación con el cliente, los desarrolladores y los gerentes. Adicionalmente, agregar más desarrolladores en proyectos ya avanzados no siempre resuelve el problema.

Ejemplo ilustrativo de programación en parejas:

Ilustración 12 Proceso de la programación en parejas.

La programación en pareja (pair programming en inglés) es una técnica empleada en el desarrollo ágil de software, consistente en trabajar en el mismo equipo dos programadores de forma conjunta. Uno de ellos (el conductor) escribe el código, mientras que el otro (observador) lo supervisa. Ambos programadores van alternando estos roles.

El código desarrollado por programadores en pareja es más corto y de mayor calidad que el que realizarían de forma individual, porque el rol de observador permite la reconsideración y mejora continua de la estrategia en la dirección del trabajo y de mejoras sobre el código que el conductor va desarrollando.

Conclusión:

Este capítulo explica la fase en donde se transcribe el código fuente, haciendo uso de prototipos, así como de pruebas y ensayos para evitar tener errores y de tenerlos, lograr corregirlos a tiempo. Independientemente del lenguaje de programación que se maneje y su versión; se crean las bibliotecas y componentes que se reutilizarán dentro del mismo proyecto para hacer proceso mucho más rápido de la programación.

CAPÍTULO 7: FASE DE PRUEBAS

7.1. Fase de pruebas

En el proceso formal de pruebas se suelen tener confusiones con mucha facilidad en los niveles de pruebas con los tipos de prueba, y a pesar de que estas estén íntimamente relacionadas, tienen significados diferentes durante su proceso. Para entender esta fase, se parte del hecho de que las pruebas pueden ejecutarse en cualquier punto del proceso de desarrollo del sistema; es aquí en donde los niveles de prueba permiten entender con mayor visión los diferentes puntos o etapas en donde pueden ejecutarse ciertos tipos de prueba. Dado esto, es común que algunos usuarios se refieran a los niveles de pruebas o intenten clasificarlos como: pruebas de desarrollador, pruebas funcionales y pruebas de usuario final. Pero, en otras palabras, la terminología que debe ser la apropiada para referirse a los diferentes niveles corresponde a las siguientes cuatro clasificaciones: pruebas unitarias, pruebas de integración, pruebas de sistema y pruebas de aceptación. Dentro de cada uno de estos niveles de prueba, se realizarán diferentes tipos de prueba tales como: pruebas funcionales, no funcionales, de arquitectura y asociadas el cambio de los productos.

A continuación, se explicará cada una de las sub etapas que contiene esta fase:

7.1.1. Pruebas Unitarias o de Componente

Este tipo de pruebas se realizan comúnmente por el equipo de desarrollo; de manera general, consisten en la ejecución de actividades las cuales permiten verificar al desarrollador que los componentes unitarios están codificados bajo condiciones de robustez, es decir, que están soportando el ingreso de datos erróneos o inesperados y demostrando así la capacidad de tratar errores de manera controlada. Además, estas pruebas de componente, suelen ser conocidas como pruebas de módulos o pruebas de clases, siendo esta, definida por el lenguaje de programación la que influye en el término a utilizar. También es importante que toda la funcionalidad de cada componente unitario sea cubierta, por al menos, dos casos de prueba, los cuales deben centrarse en probar al menos una funcionalidad positiva y una negativa.

7.1.2. Pruebas de Integración

Este tipo de pruebas se ejecutan por el equipo de desarrollo y consisten en la comprobación de los elementos del software que interactúan entre sí, y si funcionan de manera correcta.

7.1.3. Pruebas de Sistema

Las pruebas de sistema se deben ejecutar idealmente por un equipo de pruebas al que no pertenezcan el equipo de desarrollo, una buena técnica en este punto se debe a la tercerización de esta responsabilidad. La obligación del equipo encargado,

consiste en ejecutar las actividades de prueba en donde se debe verificar que la funcionalidad total de un sistema el cual fue implementado de acuerdo a los documentos de especificación definidos en el proyecto. Los casos de prueba que se diseñan en este nivel, cubren aspectos funcionales y no funcionales del sistema. Para el diseño de los casos de prueba, el equipo debe utilizar como bases de prueba, entregables, tales como: requerimientos iniciales, casos de uso, historias de usuario, diseños, manuales técnicos y de usuario final, etc. Finalmente, es importante que los tipos de pruebas ejecutados en este nivel se desglosen en un ambiente de pruebas/ pre-producción cuya infraestructura y arquitectura sea similar al ambiente de producción, evitando en todos los casos utilizar el ambiente real del cliente, debido a que se puede ocasionar fallos en los servidores, lo que provocaría la indisponibilidad en otros servicios alojados en este ambiente.

7.1.4. Detección y corrección de errores

Al momento de detectar un error ("bug"), éste debe ser corregido de manera inmediata, además, se deben tener precauciones para que errores similares no vuelvan a ocurrir. Asimismo, se generan nuevas pruebas para verificar que el error haya sido resuelto.

7.1.5. Metodología de pruebas

Las etapas de la seguridad de calidad de un sistema por lo general se dividen en su componente analítico, o sea en pruebas estáticas y dinámicas. Una forma en la que se diferencian entre ellos, es que en las pruebas estáticas se dedican solamente a evaluar la calidad con la que se está generando la documentación del proyecto, mediante revisiones cada cierto periodo de tiempo, en cambio, las pruebas dinámicas, necesitan de la ejecución del sistema para con esto medir el nivel de calidad con la que este fue codificado y el nivel de cumplimiento en relación con la especificación del sistema. [42]

Establecer pruebas dinámicas a un sistema, en la mayoría de los casos es confundido con una simple actividad de ejecución de pruebas y reporte de incidencias, pero, para productos complejos en adelante, lo recomendable es implementar de manera formal una metodología de pruebas que se ajuste y acople uniformemente con la metodología de desarrollo.

Para procesos de desarrollo basados en la metodología RUP [3] o métodos tradicionales, implementar una metodología de pruebas es totalmente viable, teniendo en cuenta esto, en este documento se establece metodologías orientadas a la documentación y a la formalización de todas las actividades ejecutadas. Y si, por el contrario, la firma desarrolladora guía su proceso bajo lineamientos basados en metodologías ágiles, "será necesario reevaluar la conveniencia de ejecutar todas las actividades que implica un proceso de pruebas formal, lo que, en la mayoría de los

casos, conlleva a reducir al mínimo las actividades relacionadas con un proceso de pruebas, circunstancia que naturalmente puede desencadenar en la liberación de productos con bajos niveles de calidad.” [42]

El proceso de pruebas formal, está compuesto, por las siguientes 5 típicas etapas:

- Planeación de pruebas.
- Diseño de pruebas.
- implementación de pruebas.
- Evaluación de criterios de salida.
- Cierre del proceso.

7.2. Pruebas de aceptación

De manera independiente de la tercerización del proceso de pruebas, la firma responsable de estas actividades emitido un certificado de calidad sobre el sistema objeto de prueba; es indispensable que el cliente designe a personal que haga parte de los procesos de negocio para la ejecución de pruebas de aceptación, es incluso recomendable, que los usuarios finales que participen en este proceso, sean independientes al personal que apoyó el proceso de desarrollo. Cuando las pruebas de aceptación son ejecutadas en instalaciones o ambientes proporcionados por la firma desarrolladora se les denominan pruebas Alpha, cuando son ejecutadas desde la infraestructura del cliente se les denomina pruebas Beta.

De una manera más sencilla de entender, Esta fase es utilizada en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada.

Las pruebas de aceptación son consideradas como “pruebas de caja negra” (“Black box system tests”). Los clientes se encargan de verificar que los resultados que observan de estas pruebas sean correctos con los que se ha pedido.

De la misma manera, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución.

Una historia de usuario no se puede considerar finalizada mientras esta en el proceso de verificación en todas las pruebas de aceptación. Dado que la responsabilidad es grupal, es recomendable publicar los resultados de las pruebas de aceptación, de manera que todo el equipo esté al tanto de esta información. [41]

7.3. Análisis de benchmarking

En esta se procura, antes de presentar la primera iteración, se deberá analizar el plan de implementación para poder llevarlo a cabo. Habrá de tenerse en cuenta los costos que la misma va a significar, los ajustes estructurales que pueda conllevar y el impacto en la cultura de la organización.

Se debe emitir informes regularmente sobre los avances y posibles dificultades encontradas. Confirmar que los recursos se están utilizando adecuadamente, y que el plan se desarrolla de acuerdo con los tiempos estimados. Si no fuera así, identificar rápidamente acciones de ajuste.

- Revisar con regularidad las mediciones del progreso que tienen un impacto directo en los cambios esperados.
- Establecer si las metas planteadas están previstas en el cronograma.
- Informar sistemáticamente los progresos obtenidos que ayuden al plan estratégico de la empresa y puedan apoyar el avance en la consecución de mejores niveles de desempeño.

7.3.1. Verificar y Ajustar Regularmente la Estrategia de Implementación en el proyecto

Se debe tener en cuenta que pocos procesos de implementación resultan como se habían planificado, o por lo menos llevándolo a la práctica, se aprende que siempre se debe contar con un cierto margen de maniobra, o plan de contingencia para direccionar determinado proceso sobre la marcha, debido a esto, se debe tener en cuenta:

- Realizar un análisis periódico de sus fuentes de Benchmarking que le permita identificar si hay nuevas innovaciones o mejores prácticas objeto de análisis. Debido a que “La innovación nunca se estanca y quien es el mejor hoy, tal vez ya no lo sea mañana”.
- Determinar las variables de ajustar sus metas de desempeño superior cuestionándose si hay nuevos competidores en la industria de los cuales se pueda usted aprender.
- Proponer nuevos cambios si fuese necesario.

7.3.2. Identificar Nuevas Oportunidades para Nuevos Procesos de Benchmarking

Una vez teniendo el estudio completo y los cambios puestos en marcha, no se debe de olvidar que el proceso de búsqueda de mejorar nunca termina, en la medida que la innovación tampoco se detiene; debido a que las necesidades y expectativas de los

usuarios se transforman o se afectan por los cambios en las organizaciones.

Se debe de considerar lo siguiente:

- Mantener la búsqueda de la excelencia con la tensión permanente de apostar por la aplicación de la innovación.
- Sostener la tendencia de incorporar siempre las mejores prácticas en sus procesos.
- Anticiparse a las expectativas o requerimientos de sus usuarios, que aún no se hayan detectado.

Ejemplo de fases de pruebas: Pruebas de aceptación, casos de prueba

Tomando como ejemplo un sistema Escolar en línea, En la Tabla 19 se definen de forma general las pruebas de aceptación y en las tablas 20-23 se describen cada una de ellas, las cuales fueron utilizadas para la primera iteración.

Tabla 19: Pruebas de aceptación.

Número de la prueba	Número de la historia	Nombre de la prueba
1	1	Acceso al sistema
2	2	Creación de Permiso
3	3	Gestión de Usuarios
4	4	Registro Docentes

Tabla 20: Caso de prueba de acceso al sistema.

CASO DE PRUEBA	
CÓDIGO: **1**	Nº Historia de usuario: 1
Historia de usuario: Acceso al Sistema	
Condiciones de Ejecución: **Cada usuario debe contar con un perfil de usuario y su contraseña para poder accesar a las funciones del sistema de acuerdo a su rol.**	
Entrada/Pasos de Ejecución: Dar clic en el enlace sesión Llenar el formulario usuario introduciendo su nombre de usuario y contraseña Luego pulsar el botón INICIAR SESION	
Resultado Esperado: **Acceso a las funcionalidades del sistema dependiendo del tipo de usuario y el rol que desempeña en el mismo.**	
Evaluación de la Prueba: La prueba se concluyó satisfactoriamente	

Tabla 21: Caso de prueba creación de permiso.

CASO DE PRUEBA	
CÓDIGO: **2**	Nº Historia de usuario: 2
Historia de usuario: Creación de Permisos	
Condiciones de Ejecución: **El administrador tendrá que iniciar sesión en el sistema y posteriormente seleccionar la opción de DEFINIR ROLES DE USUARIOS**	
Entrada/Pasos de Ejecución: Llenar el formulario correspondiente a la definición de los roles o permisos del sistema. Posteriormente presionar en el botón GUARDAR	
Resultado Esperado: Registro de roles de usuarios almacenados satisfactoriamente	
Evaluación de la Prueba: La prueba se realizó satisfactoriamente	

Tabla 22: Caso de prueba gestión de usuario.

CASO DE PRUEBA	
CÓDIGO: **3**	Nº Historia de usuario: 3
Historia de usuario: Gestión de usuarios	
Condiciones de Ejecución: **El administrador del sistema o el usuario que desea cambiar las configuraciones de su cuenta tendrá que autenticarse primero para poder ingresar al mismo.**	
Entrada/Pasos de Ejecución: Cada usuario con acceso al sistema, si requiere hacer alguna modificación desde su perfil tendrá que seleccionar la opción EDITAR Luego tendrá que llenar el formulario correspondiente introduciendo su contraseña actual Posteriormente definir un nuevo indicio de contraseña	
Resultado Esperado: **Cuenta de usuario actualizada correctamente**	
Evaluación de la Prueba: La Prueba finalizó con éxito	

Tabla 23: Caso de prueba registrar docente.

CASO DE PRUEBA	
CÓDIGO: **4**	Nº Historia de usuario: 4
Historia de usuario: Registrar Docentes	
Condiciones de Ejecución: **El Administrador deberá estar con sesión iniciada en el sistema**	
Entrada/Pasos de Ejecución: Selecciona la pestaña administración, la opción REGISTRAR DOCENTES Cargar los docentes por departamento Luego seleccionar la opción GUARDAR	
Resultado Esperado: La información del Registro de los Docentes guardados	
Evaluación de la Prueba: La prueba finalizó correctamente	

Como resultado de entrega de esta iteración del sistema, el cliente quedó satisfecho con las funcionalidades de los módulos que se desarrollaron; cabe destacar que sí se solicitan cambios, estos serán una prioridad fundamental en la siguiente iteración.

Conclusión:

Este capítulo toma en cuenta a los procesos que ya han sido programados después de haber estructurado el sistema y se comprueba que funciona correctamente y que cumple con los requisitos, además de dar a conocer las técnicas que se deberán utilizar para establecerse como un mejor sistema de proyecto a comparación de otro, mediante la técnica de benchmarking; antes de ser entregado al cliente.

CAPÍTULO 8: FASE DE CLAUSURA

8.1. Fase de clausura

Esta fase suele ser la fase más corta de un proyecto, aun así, no debe ser considerada la menos importante a comparación con las otras. Debido a que en esta fase se establece el cierre formal del proyecto, revisa los éxitos y los fracasos con miras a mejorar el próximo. Esta fase puede ser usada para comunicar el archivo del proyecto, captura de lecciones aprendidas y evaluación de la ejecución contra el plan del proyecto.

El valor obtenido al realizar una óptima finalización de un proyecto es el aprovechamiento de toda la información y la experiencia adquirida. Debido a esto, un cierre inadecuado podría traer consecuencias que afecten la estrategia corporativa de la empresa.

Realizar el cierre de un proyecto implica un conjunto de procesos, como lo es el inicio, la planificación, la ejecución o el control. Según el PMBOK [43] el cierre de proyectos incluye dos procesos:

- ***Cerrar el proyecto o la fase del proyecto:*** contiene elementos como la terminación de los detalles técnicos faltantes del proyecto. También se considera la auditoría de los trabajos para asegurar que están completos. Seguidamente debe realizarse la transferencia del producto, servicio o resultado final. Tras ello vienen las actualizaciones de los documentos del proyecto y, finalmente, la transferencia oportuna de los recursos. [44]
- ***Cerrar las adquisiciones:*** este proceso contiene las auditorías de las adquisiciones, los acuerdos negociados y las actualizaciones a los activos de los procesos de la organización. [44]

Muchos factores influyen en que los trabajos finales se realicen sin motivación alguna, y por lo tanto sin rendimiento. Por ello cabe destacar la importancia de escoger buenos equipos dedicados al cierre de proyectos. [44]

Es de alta importancia implantar políticas en la empresa, que establezcan a cabalidad para que se cumplan los procedimientos de cierre, a pesar de las circunstancias que obliguen a salir con rapidez de un proyecto. [44]

El primer proceso es parte del área de Integración mientras que el segundo es parte de adquisiciones. Estos procesos pueden ser aplicados a todas las actividades de cierre que se pueden suscitar de manera repetida en las diferentes etapas del proyecto.

Pero, sin duda, el cierre más importante y a su vez el que más dejan de lado, corresponde a su finalización y tiene lugar en la etapa final de su ciclo de vida. Al cierre del proyecto, el director del proyecto revisará toda la información anterior procedente de los cierres de las fases previas y con esto asegura que todo el trabajo del proyecto está completo y que el proyecto ha alcanzado sus objetivos.

Así el cierre de proyectos requiera de dos procesos estandarizados y esta sea la fase más sencilla, las actividades que se deben realizar tienen un cierto grado complejidad muy particular y a veces no logra poseer todo el apoyo del equipo, u otros inconvenientes que se suelen presentar, debido a que la parte de desarrollo del proyecto ya ha concluido. El listado que se indica más abajo, sería aplicable en ciertos proyectos donde obviamente existe un cliente externo a la empresa que ejecuta el proyecto.

Al momento en que se genera el cronograma del proyecto, también deberían planificarse las actividades que deben llevarse a cabo para cerrar de manera correcta el proyecto. Estas actividades incluyen:

En la culminación del proyecto se debe mantener una reunión de revisión. Una reunión que se realizará con las partes interesadas y el equipo del proyecto para concluir formalmente el proyecto. Esta reunión incluirá un resumen del proyecto, la documentación de cosas que salieron bien y cosas que salieron mal, fortalezas y debilidades del proyecto y los procesos de gestión de proyectos, y los pasos restantes necesarios para terminar el proyecto. Las técnicas o procesos que funcionaron especialmente bien, o mal, se identifican como las principales enseñanzas del proyecto. Si su organización tiene una manera de publicar o aprovechar estos aprendizajes fundamentales, deben ser enviados al grupo apropiado. (Aprendizajes clave que parecen funcionar consistentemente en muchos proyectos, en muchas circunstancias, podría ser elevado a la categoría de mejores prácticas y se utilizará para todos los proyectos similares.) [45]

Esta fase se centra en lo que el proyecto iba a lograr y lo que el proyecto efectivamente realizó. La discusión debe conducir a un conjunto de aprendizajes clave que describen lo que salió bien y lo que no funcionó (Dado el caso).

Declarar el éxito o el fracaso: En gran parte de los casos es obvio identificar que el proyecto fue un éxito total y en otros casos el proyecto es un fracaso total. Sin embargo, en muchos casos, se han producido resultados de ambas características. Por ejemplo, los principales aportes hayan sido realizados, pero el proyecto sobrepasó su presupuesto. O bien, el equipo del proyecto finalizó a tiempo y dentro del presupuesto, pero la solución solo alcanzó el 70% de los requerimientos. Para esto, es necesario la declaración para definir por adelantado cuáles son los criterios de éxito. Si se alcanza un acuerdo las partes interesadas significa el éxito y el equipo

del proyecto puede evaluar en relación con esos criterios

Transición de la solución a las operaciones de apoyo (si corresponde). Si la solución va a continuar fuera del proyecto, debe efectuarse una transición adecuada a la organización con el apoyo suficiente. La transición incluye la transferencia de conocimientos al equipo de soporte, la transferencia completa de toda la documentación, la transferencia de cualquier trabajo remanente. [45]

8.1.1. Presentar entregables

Establecer los entregables tal como fueron relacionados en los planes de proyecto y calidad y su estado final (completado, abandonado, pospuesto, cerrado).

Cambios de alcance aprobados durante el curso del proyecto

Relacionar los cambios de alcance aprobados durante el lapso del desarrollo del proyecto y describir su impacto sobre el proyecto. Las actas de su aprobación (comité de cambios) pueden ser un anexo del entregable de cierre.

Actividades de control de calidad

Se debe incluir una tabla resumen de las actividades de control de calidad

asociadas a cada entregable en el plan de calidad y para cada una establecer si fueron llevadas a cabo y una breve descripción del beneficio.

Performance de ejecución de las actividades del programa

En forma breve debe exponerse como fue el cumplimiento del programa con respecto a lo planeado. Para aquellas actividades del plan que presenten varianzas significativas, debe explicarse razón, plan de choque que se realizó y efecto sobre el proyecto de manera general. Un buen nivel de síntesis es una tabla que incluya actividades principales, fecha planeada, fecha real y explicación de varianzas con el alcance aquí expresado.

Performance de ejecución del presupuesto

Al igual que en caso anterior, de forma breve se debe exponer como fue la ejecución del presupuesto con respecto a lo planeado. Para aquellas actividades o rubros del plan que presentaron varianzas significativas, se debe explicarse con un gran grado de detalle los motivos de tales varianzas. Un buen nivel de síntesis es una tabla que incluya actividades/rubro, valor presupuestado, valor ejecutado y explicación de varianzas con el nivel de cobertura expuesto.

Impacto sobre el servicio

Si el proyecto tuvo impacto en la prestación del servicio, se debe explicar en qué forma fue impactado y qué manejo se realizó del impacto.

Evolución de supuestos y riesgos

Se debe relacionar los supuestos entregables en el Plan de Proyecto y establecer si fueron correctos y cómo fueron manejados.

También se debe relacionar del documento de Plan de Riesgos solamente aquellos riesgos que terminaron impactando el proyecto y como fue manejado su impacto (en teoría si las medidas de contingencia estuvieron suficientemente estudiadas, tendría que ser lo realizado).

Acuerdos para soporte y evolución

Es necesario describir brevemente los acuerdos para el soporte en curso, referenciando cuando sea necesario un acuerdo de nivel de servicio (service level agreement) y el resultado del traspaso a los usuarios y la organización de prestación de servicios. [46]

Reconciliación del presupuesto del proyecto

Se expone el estado de las cuentas del proyecto, describiendo cuando sea necesario los acuerdos sobre la gestión de operaciones pendientes.

Lecciones aprendidas del proyecto

Explicar qué lecciones han sido aprendidas de este proyecto y cómo pueden ser comunicadas y aplicadas. Conocimiento que es legado de este proyecto a la empresa.

Acciones y responsables

Los procesos y acciones que deben ser llevados a cabo, responsables de su cumplimiento y fecha planeada de terminación:

- Completar entregables.
- Hacer frente a las cuentas del proyecto.
- Asegurar que los beneficios completos del proyecto sean alcanzados.
- Comunicar y aplicar las lecciones aprendidas.

8.2. Evaluación iteral del proyecto

La iteración es un conjunto de tareas ajustadas en el tiempo que se centran estrechamente en producir un ejecutable o presentable. Para todas las iteraciones, excepto la última iteración de transición, es un producto intermedio, producido para centrar la atención en mitigar el riesgo y dirigir el proyecto a una entrega satisfactoria. Al ser enfocado en un entregable, este fuerza una integración prácticamente continua y permite al proyecto atender a los riesgos técnicos en la fase inicial, a la vez que se reducen los riesgos de los asistentes. [47]

La Evaluación iteración implica una determinada cantidad de revisión (de los productos de trabajo existentes), así como un cambio en la revisión.

En resumen, es necesario de una determinada cantidad de revisiones para dar por finalizado un producto el cual se vaya a entregar y hacerlo con calidad, esto se logra conseguir mediante la creación de productos intermedios y la evaluación de la idoneidad de la arquitectura del producto al principio y regularmente, la calidad del producto final aumenta, a la vez que los cambios

8.2.1. Pasos para realizar la evaluación iteral del proyecto

8.2.1.1. Determinar el ámbito de iteración

La iteración y su aplicación se basa en cuatro factores:

- Los mayores riesgos del proyecto
- La manera de funcionar del sistema
- La designación del tiempo a cada iteración del plan de proyecto
- La fase y sus objetivos específicos

Al inicio de toda iteración, se selecciona suficiente trabajo para cubrir el tiempo planificado previamente para la iteración; aunque la persona encargada de gestionar los proyectos tiene permitida una mayor flexibilidad para responder a las restricciones de recursos y otras consideraciones tácticas durante el desarrollo del plan de iteración. De igual manera, el trabajo planificado para las iteraciones que no se hayan completado (debido a varios factores como la reducción de tiempo para cumplir la planificación) tendrá una mayor prioridad.

El área del trabajo también debe basarse en un enfoque sensible en cuanto al personal que se posee y se puede utilizar para completarlo durante la iteración.

Ej. *"Normalmente no se puede duplicar el trabajo completado en una iteración duplicando el personal que se le aplica, aunque dichos recursos estén disponibles. La cantidad aproximada de personal que se puede aplicar de forma eficaz viene determinada por el tamaño de software y la arquitectura general, y se puede determinar mediante modelos de cálculo como, por ejemplo, COCOMO II"*

La manera en cómo se ejecuta una iteración se puede realizar gestionando a través de límites de tiempo; esto es, el ámbito y la calidad se gestiona activamente para cumplir la fecha final.

En la fase de elaboración:

Existen tres controladores principales para definir los objetivos de una iteración en elaboración:

- Riesgo
- Gravedad
- Cobertura

El principal indicador para definir los objetivos de iteración son los riesgos. Se debe calmar o retirar los riesgos tan pronto como se pueda. Por lo general este indicado siempre es el caso en la fase de elaboración, debido que es donde la mayoría de los riesgos se deben mitigar, aunque este puede continuar siendo un elemento clave en la construcción ya que algunos riesgos permanecen altos o se descubren nuevos riesgos. Debido a que el objetivo de la fase de elaboración es crear una línea base de la arquitectura, se deben tener en cuenta otras consideraciones como, por ejemplo, asegurarse de que la arquitectura engloba todos los aspectos del software que se va a desplegar (cobertura). Esto tiene gran relevancia, debido que la arquitectura se utilizará en la planificación adicional: organización del equipo, cálculo del código que se va a desarrollar, etc.

Para finalizar, aunque centrarse en los riesgos es importante, siempre se debe tener en mente cuáles son las principales misiones del sistema; dar solución a todos los problemas graves es bueno, pero no se debe hacer en deterioro de la funcionalidad central: hay que tomar precauciones de que las funciones y los servicios críticos del sistema estén cubiertos (gravedad), aunque no se perciba ningún riesgo asociado con ellos.

En la lista de **riesgos**, para los riesgos más graves, identifique algún caso de ejemplo en un guion de uso que obligue al equipo de desarrollo a "confrontar" el riesgo.

Ej.

- *Si existe un riesgo de integración como, por ejemplo, "el trabajo correcto de la base de datos ", asegúrese de incluir un caso de ejemplo que implique alguna interacción de la base de datos, aunque sea muy modesta.*
- *Si existe un riesgo de rendimiento como, por ejemplo, "el tiempo para calcular la respuesta del sistema en la web", asegúrese de tener un caso de ejemplo que incluya este cálculo, al menos para el guion más obvio y frecuente.*

En cuanto a la **gravedad**, hay que tener en cuenta de que se incluyan las funciones o servicios fundamentales proporcionados por el sistema. Se debe seleccionar un caso de ejemplo del guion de uso que represente la forma más común y frecuente del servicio o la característica que ofrece el sistema.

Ej. *"Para un computador, la información que es almacenada de lugar a lugar es el requisito obvio para una de las primeras iteraciones (Debido a la perdida que se pueda ocasionar). Es mucho más importante que solucione esto que las modalidades de anomalías complejas en la configuración del programador del sistema de manejo de errores."*

En cuanto a la **cobertura**, al final de la fase de elaboración, se debe incluir casos de ejemplo dedicados a áreas que sabe que van a necesitar desarrollo, aunque no sean graves ni arriesgadas.

A menudo, resulta económico crear casos de ejemplo largos, de extremo a extremo, que engloben varios problemas a la vez.

El riesgo es crear casos de ejemplo demasiado "gruesos", es decir, intentar cubrir demasiados aspectos y variantes diferentes, y demasiados casos de error (consulte Plan de iteración)

De igual manera, en la fase de **elaboración**, se debe tener en cuenta que algunos de los riesgos pueden ser de naturaleza más humana o programática (cultura del equipo, formación, selección de herramientas, nuevas técnicas, etc.) y que pasar por la iteración permite mitigar estos riesgos.

Ej.

- ***Crear un registro de usuarios en una estación de trabajo cliente para almacenarlo en la base de datos del servidor, que incluya un diálogo de usuario, pero sin incluir todos los campos, suponiendo que no se ha detectado ningún error.***

  Combina funciones críticas con riesgos de integración (base de datos y software de comunicación) y problemas de integración (tratar con dos plataformas diferentes). Asimismo, obliga a los diseñadores a familiarizarse con la nueva herramienta de diseño de GUI. Por último, produce un prototipo que se puede demostrar al usuario para obtener información de retorno.

- ***Asegurarse de que se pueden crear gran cantidad de usuarios, y que el acceso a uno no tarde demasiado tiempo.***

  Soluciona algunas necesidades clave de rendimiento (volumen o datos, y tiempo de respuesta) que pueden afectar gravemente a la arquitectura si no se cumplen.

- ***Deshacer un cambio de dirección de usuario.***

  Una característica sencilla que obliga a los diseñadores a pensar un diseño de todas las funciones "deshacer". Por su parte, esto puede provocar alguna retrotracción en los usuarios sobre qué se puede deshacer con un coste razonable.

- ***Completar todos los guiones de uso relativos a la gestión de la cadena de suministros.***

  El objetivo de la fase de elaboración es completar la captura de requisitos, quizás también conjunto a conjunto.

En la fase de desarrollo:

Cuando el proyecto pasa a la fase de desarrollo, los riesgos continúan siendo un controlador clave, sobre todo cuando se descubren nuevos riesgos inesperados. Pero la completitud del guion de uso también empieza a ser un controlador. Las iteraciones se pueden planificar característica a característica, intentando completar algunas de las más importantes al principio para que se puedan probar ampliamente durante más de una iteración. Finalizando el desarrollo, el principal objetivo será la solidez de los guiones de uso completos.

En la fase de cierre:

El principal objetivo es finalizar la generación del sistema. Los objetivos de una

iteración se establecen en términos de qué errores se solucionan, y qué mejoras de rendimiento o utilización se incluyen. Si se tienen que eliminar (o inhabilitar) características para llegar a tiempo al final de la construcción (objetivo de IOC o "beta"), pueden quedar sin completar o activar si no ponen en peligro lo que se ha conseguido hasta ahora.

En resumen, para definir el contenido de una iteración, necesitará:

- el plan de proyecto
- el estado actual del proyecto (bajo seguimiento, retrasos, número elevado de problemas, arrastra los requisitos, etc.)
- una lista de casos de ejemplo o casos de uso que se deben completar al final de la iteración
- una lista de riesgos que se deben solucionar al final de la iteración
- una lista de cambios que se deben incorporar al producto (arreglos de errores, cambios en los requisitos)
- una lista de las clases principales o paquetes que deben estar completamente implementados

Estas listas deben disponer de un rango. Los objetivos de una iteración deben ser agresivos para que, cuando surjan dificultades, los elementos se puedan eliminar desde las iteraciones basándose en sus rangos. [48]

Ejemplo de entregables

A manera de ejemplo se refleja una sección de la tabla en mención incluida en el entregable de un proyecto real.

Tabla 24: Entregable del proyecto.

Entregable	Estado Final
Plan de instalación del sistema	Cerrado
Sistema de información configurado	Cerrado
Plan de migración de datos	Cerrado

Se debe Indicar qué medidas se han puesto en marcha para completar entregables que quedan pendientes, en caso de ser aún requeridos.

Ejemplo de entregables de control de calidad

A manera de ejemplo se refleja una sección de la tabla en mención incluida en el entregable de un proyecto real (note que en el plan de calidad aparece una tabla

por cada entregable, aquí solo se incluye una única tabla con todos los entregables)

Tabla 25: Entregables de control de calidad.

Entregable	Actividad de control de calidad
Sistema de información configurado, implementado libre de errores	El DBA validó número, tipo y estado de objetos de la Base de Datos. Se verificó el ingreso exitoso al sistema desde una estación cliente. Se validó cumplimiento de lista de verificación de la instalación.
Documento de especificación de datos requeridos por la solución	Se validó que la información requerida por SINU fuese coherente con el modelo de datos del aplicativo. Se verificó claridad de la semántica propia de cada dato o metadato. Se chequeó que la información requerida por SINU soportase la implantación de los procesos de negocio que adelantan la academia y la administración.

Conclusión:

Este capítulo se puede considerar el final, debido a que la fase de clausura, de por finalizado el proyecto de trabajo sino se toma en cuenta al Inbound Marketing (El cual es opcional en esta metodología). Esta fase compone el proceso de gestión del mismo, y aplica tanto al proyecto en general como a cada una de las fases de su ciclo de vida. Teniendo así y de esta manera, tenemos un proyecto que se ejecuta en fases. Cada una de las fases debe incluir su proceso de aceptación y cierre, ajustado a sus características concretas, teniendo en cuanto el tiempo que se va a establecer para presentar los entregables en las iteraciones.

REFERENCIAS BIBLIOGRÁFICAS

[1] R. M. A. Hintelholher, "Identidad y diferenciación entre Método y Metodología," [Online]. Available: http://www.redalyc.org/pdf/4264/426439549004.pdf. [Accessed 06 02 2018].

[2] R. Pressman, Software Engineering: a practitioner's approach, Mc Graw Hill, 2005.

[3] IBM, "IBM Rational Unified Process (RUP)," [Online]. Available: https://www-01.ibm.com/software/rational. [Accessed 18 02 2017].

[4] K. Beck, "Extreme programming (XP): a gentle introduction," [Online]. Available: http://www.extremeprogramming.org/. [Accessed 15 04 2017].

[5] S. group, ""Scrum," [Online]. Available: https://www.scrum.org/. [Accessed 20 04 2017].

[6] OOHDM, "OOHDM model," [Online]. Available: http://www.hipertexto.info/documentos/oohdm.htm. [Accessed 22 04 2017].

[7] K. W. a. J. Beatty, Software Requirements, Third Edition, Washington: Microsoft Press, 2013.

[8] E. Gottesdiener, Marzo 2008. [Online]. Available: https://pdfs.semanticscholar.org/86fb/f068cb46577677b476b4cfef8d0c83c172ec.pdf. [Accessed 02 Marzo 2018].

[9] J. MONTILVA, "Modelado de Negocios: del Espacio del Problema al Espacio de la Solución," In Ideas, vol. VII, 2007.

[10] O. PASTOR, A. MARTÍNEZ and H. U. G. O. ESTRADA, "Generación de Especificaciones de Requisitos de Software a partir de Modelos de Negocios," Gerencia Tecnológica e Informática, vol. II, no. 1, pp. 53-65.

[11] J. MONTILVA and J. BARRIOS, "Modelado de Negocios.," Centro de Excelencia en Ingeniería del Software, vol. II, 2007.

[12] A. MARTÍNEZ, H. ESTRADA and O. PASTOR, El modelo de negocio como origen de especificaciones de requisitos de software: una aproximación metodológica, Puebla, 2002.

[13] J. R. P. e. a. CURTO, BPM (Business Process Management): Cómo alcanzar la agilidad y eficiencia operacional a través de BPM y la empresa orientada a procesos., BPMteca. com, 2013.

[14] O. M. L. LEÓN and J. A. A. ESPAÑA, "La Importancia del Modelado de Procesos de Negocio como Herramienta para la Mejora e Innovación.," Revista Raites, vol. IV, no. 7, pp. 61-72., 2009.

[15] H. M. A. P. O. &. S. J. Estrada, "Generación de Especificaciones de Requisitos de Software a partir de Modelos de Negocios: un enfoque basado en metas.," V Workshop de Engenharia de Requisitos, vol. II, no. 11, 2002.

[16] F. PINCIROLI and L. ZELIGUETA, Modelado de negocios orientado a aspectos con AOP4ST., Buenos Aires: ITBA, 2017.

[17] J. A. SENN, E. G. U. MEDAL and O. A. P. VELASCO, Análisis y diseño de sistemas de información, McGraw-Hill, 1992.

[18] V. F. ALARCÓN, Desarrollo de sistemas de información: una metodología basada en el modelado., Univ. Politèc. de Catalunya,, 2006.

[19] G. A. RIVAS, Auditoría informática, Ediciones Díaz de Santos, 1989.

[20] R. DUBS DE MOYA, "El proyecto factible: una modalidad de investigación.," Sapiens. Revista Universitaria de Investigación, vol. III, no. 2, 2002.

[21] N. R. MEAD, "Requirements management and requirements engineering: You can't have one without the other.," Cutter IT Journal, vol. XIII, no. 5, 2000.

[22] A. D. TORO and B. B. JIMÉNEZ, "Metodología para la elicitación de requisitos de sistemas software. Informe Técnico LSI-2000-10," Facultad de Informática y Estadística Universidad de Sevilla, 2000.

[23] M. ARIAS CHAVES, "La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software.," InterSedes, vol. VI, no. 10, 2005.

[24] A. ORJUELA DUARTE and M. ROJAS, "Las metodologías de desarrollo ágil como una oportunidad para la ingeniería del software educativo.," Revista Avances en Sistemas e Informática, vol. V, no. 2, 2008.

[25] G. D. S. HADAD, "so de Escenarios en la Derivación de Software," Facultad de Ciencias Exactas, 2008.

[26] M. d. l. A. S. Á. LÓPEZ, "Áncora: Análisis de requerimientos de software conducente al reuso de artefactos.," Universidad Veracruzana, Veracruz, 2006.

[27] J. GREMBI, Secure Software Development: A Security Programmer's Guide., Course Technology, 2008.

[28] M. MANIES and U. NIKUAL, "La elicitación de requisitos en el contexto de un proyecto software.," USBMed, vol. II, no. 2, pp. 25-29., 2011.

[29] N. D. DÁVILA, "Ingeniería de requerimientos: Una guía para extraer, analizar, especificar y validar los requerimientos de un proyecto.," Universidad ORT.

[30] M. I. e. a. LUND, "CUPIDo-Plantilla para documentar casos de uso.," in V Congreso de Tecnología en Educación y Educación en Tecnología., 2010.

[31] IBM, "IBM," 2007. [Online]. Available: https://www.google.com/

[32] A. e. a. DURÁN TORO, Un entorno metodológico de ingeniería de requisitos para sistemas de información, 2000.

[33] P. y. K. G. SAWYER, "Software Engineering Book Of Knowledge (SWEBOK)," 2014. [Online]. Available: https://www.computer.org/web/swebok. [Accessed 2018 Marzo 2].

[34] M. J. ESCALONA and N. KOCH, Ingeniería de Requisitos en Aplicaciones para la Web–Un estudio comparativo., Universidad de Sevilla, 2002.

[35] V. e. a. PELECHANO, OO-Method: Una apuesta por la integración de técnicas formales y semi-formales en la ingeniería de requisitos., Sevilla: I Jornadas Nacionales de Ingeniería de Software , 1996.

[36] ISOTools, "www.isotools.org," [Online]. Available: https://www.isotools.org/soluciones/evaluacion-y-resultados/benchmarking/. [Accessed 02 08 2017].

[37] L. Maram, "¿Qué es y cómo hacer benchmarking; 5 ejemplos de sustentabilidad?," [Online]. Available: https://www.luismaram.com/como-hacer-benchmarking-en-sustentabilidad/. [Accessed 27 02 2018].

[38] D. A. Silva, Construyendo aplicaciones Web con una metodología de diseño orienta a objetos, Buenos Aires: Universidad Nacional de la plata.

[39] Addison-Wesley, Design Patterns: Elements of reusable object-oriented software, E. Gamma, 1995.

[40] "CCS Test Suite, W3C Core Styles," [Online]. Available: http://www.w3.org/Style/CSS. [Accessed 01 03 2018].

[41] B. L. Yoland, Metodología Ágil de Desarrollo de Software – XP, ESPE, MEVAST, 2015.

[42] J. Z. Sánchez, "pruebasdelsoftware.com," [Online]. Available: https://pruebasdelsoftware.wordpress.com/. [Accessed 25 08 2017].

[43] PMBOOK, "La guia PMBOOK," [Online]. Available: https://uacm123.weebly.com/9-gestioacuten-de-las-adquisiciones-del-proyecto.html. [Accessed 28 02 2018].

[44] "Conexionesan," [Online]. Available: https://www.esan.edu.pe/apuntes-empresariales/2016/11/consideraciones-a-tomar-en-cuenta-al-cerrar-un-proyecto/. [Accessed 28 02 2018].

[45] A. N. Figuerola, El Cierre de los Proyectos, buenos Aires, 2008.

[46] M. Hernández, Administración de Proyectos de Software- PMI, Universidad del norte, 2010.

[47] RUP, "RUP.es," [Online]. Available: http://cgrw01.cgr.go.cr/rup/RUP.es/SmallProjects/core.base_rup/tasks/develop_iteration_plan_144CF253.html. [Accessed 28 02 2018].

[48] RUP, "rup/RUP.es," [Online]. Available: http://cgrw01.cgr.go.cr/rup/RUP.es/SmallProjects/core.base_rup/workproducts/rup_iteration_plan_623AFF7F.html. [Accessed 28 02 2018].

[49] U. P. Fabra, "TecnoCampues," [Online]. Available: http://consultoras-digitales.micd.tecnocampus.cat/benchmarking-apple-samsung-microsoft/. [Accessed 28 02 2018].

[50] J. Joskowicz, Reglas y Prácticas en eXtreme Programming, Universidad de Vigo, España., 2008.

[51] X. –. J. V. 1.2.2, "The Apache Software Foundation," [Online]. Available: http://xml.apache.org/xalan/. [Accessed 01 03 2018].

[52] M. H. Kay and A. Saxon. [Online]. Available: http://users.iclway.co.uk/mhkay/saxon/instant.html. [Accessed 01 03 2018].

[53] W. Recommendation, "World Wide Web Consortium (W3C) Extensible Stylesheet Language Transformations (XSLT)," 16 11 1999. [Online]. Available: http://www.w3.org/TR/xslt. [Accessed 01 03 2018].

[54] S. &. N. T. &. K. K. &. U. N. &. H. M. ori, "Project Management Patterns to Prevent Schedule Delay Caused by Requirements Changes - Empirical Study on a Successful Project," roceedings of the 4th International Conference on Software and Data Technologies, vol. I, pp. 115-120, 2009.

Ingeniería y Tecnología

3ciencias