



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ESCOM

Trabajo terminal

Sistema generador de documentos de casos de uso "TESSERACT"

2018-B140

Presentan

Jiménez Chávez Luis Gerardo
López Orozco Diego Efrain
Martínez Ibáñez Esteban Pablo
Olvera Neria Yamile Giselle

Directores

M. en C. Hermes Francisco
. Montes Casiano

M. en C. José Jaime
López Rabadán

5 2



noviembre del 2019



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
SUBDIRECCIÓN ACADÉMICA**



No de TT:2018-B140

noviembre de 2019

Documento Técnico

**Sistema generador de documentos de casos de uso
"TESSERACT"**

Presentan

Jiménez Chávez Luis Gerardo [1]

López Orozco Diego Efrain [2]

Martínez Ibáñez Esteban Pablo [3]

Olvera Neria Yamile Giselle [4]

Directores

M. en C. Hermes Francisco

M. en C. José Jaime

Montes Casiano

López Rabadán RESUMEN

En este reporte se presenta la documentación técnica del Trabajo Terminal 2018-B140 titulado "Sistema generador de documentos de casos de uso", cuyo objetivo es desarrollar un sistema web que asista en la generación de la documentación de casos de uso de un proyecto de software con base en una plantilla predefinida a fin de contribuir en su proceso de creación.

Palabras clave: Administración de Proyectos, Ciclo de vida del software, Documento de caso de uso, Ingeniería de Software, Lenguaje Unificado de Modelado (UML)

Correos electrónicos:

[1] lgjc1im11@gmail.com

[2] quakediego33@hotmail.com

[3] este_p@hotmail.com

[4] ggiselle124@gmail.com



ESCUELA SUPERIOR DE CÓMPUTO
SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE FORMACIÓN INTEGRAL
E INSTITUCIONAL
COMISIÓN ACADÉMICA DE TRABAJO
TERMINAL



Ciudad de México a de noviembre de 2019

DR. FLAVIO ARTURO SÁNCHEZ GARFIAS
PRESIDENTE DE LA COMISIÓN ACADÉMICA
DE TRABAJO TERMINAL
P R E S E N T E

Por medio del presente, se informa que los alumnos que integran el TRABAJO TERMINAL:
 2018-B140 titulado "Sistema generador de documentos de casos de uso TESSERACT" concluyeron
 satisfactoriamente su trabajo.

Los discos (DVD's) fueron revisados ampliamente por sus servidores y corregidos cubriendo el alcance
 y el objetivo planteados en el protocolo original y de acuerdo a los requisitos establecidos por la
 Comisión que Usted preside.

ATENTAMENTE

JOSE JAIME LÓPEZ RABADÁN

HERMES FRANCISCO MONTES CASIANO

Advertencia

“Este documento contiene información desarrollada por la Escuela Superior de Cómputo del Instituto Politécnico Nacional, a partir de datos y documentos con derecho de propiedad y por tanto, su uso quedará restringido a las aplicaciones que explícitamente se convengan.”

La aplicación no convenida exime a la escuela su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte técnico podrá obtenerse en:

La Subdirección Académica de la Escuela Superior de Cómputo del Instituto Politécnico Nacional, situada en Av. Juan de Dios Bátiz s/n Teléfono: 57296000, extensión 52000.

Índice general

1. Introducción	9
1.1. Problemática	10
1.2. Propuesta	10
1.3. Objetivos	12
1.3.1. Objetivo General	12
1.3.2. Objetivos Específicos	12
1.4. Justificación	13
1.5. Estructura del Documento	14
2. Marco teórico	15
2.1. Ingeniería de software	15
2.2. Proceso de desarrollo de software	17
2.2.1. Análisis	17
2.3. Herramienta CASE	20
2.3.1. Definición	20
2.4. Implementación del desarrollo	20
2.4.1. Patrones de diseño	20
2.4.2. Framework	23
2.5. Pruebas	24
2.5.1. Pruebas Dinámicas	25
2.5.2. Prueba de sistema	25
3. Estado del Arte	27
3.1. Antecedentes	28
3.1.1. UCD-Generator - Una aplicación LESSA para el diseño de casos de uso	28

3.1.2. Generación automatizada de diagramas de casos de uso a partir de requerimientos de usuarios	28
4. Análisis de mercado	30
4.1. Situación actual y evolución del mercado	30
4.1.1. Industria Mexicana del Software	31
5. Estimación de tiempo y costo	33
5.1. Puntos de función	33
6. Arquitectura	37
6.1. Frameworks	37
6.1.1. Spring	37
6.1.2. Struts	37
6.2. Modelo MVC	38
6.3. Arquitectura N Capas	38
6.4. Patrones de Diseño	38
6.4.1. Interface	38
6.4.2. Singleton	38
6.5. Logger log4j	39
7. Metodología	40
7.1. Sprint 0: Configuración del ambiente de trabajo	41
7.1.1. Selección de herramientas	41
7.2. Sprint 1: Gestión de colaboradores	42
7.2.1. Iniciar sesión	42
7.2.2. Gestionar colaborador	42
7.3. Sprint 2: Gestión de proyectos	43
7.3.1. Gestionar proyectos de administrador	43
7.4. Sprint 3: Gestión de Proyectos de Colaborador	44
7.4.1. Gestionar proyectos de colaborador	44
7.5. Sprint 4: Gestión de Módulos	44
7.5.1. Gestionar módulos	44
7.6. Sprint 5: Gestión de Términos de glosario	45
7.6.1. Gestionar términos del glosario	45
7.7. Sprint 6: Gestión de Entidades	46
7.7.1. Gestionar Entidades	46
7.8. Sprint 7: Gestión de Atributos	46
7.8.1. Gestionar Atributos	46

8. Pruebas Realizadas	48
8.1. Pruebas Dinámicas	48
8.1.1. Diseño de Pruebas	48
9. Bibliografía	54

Índice de figuras

2.1. Capas de la ingeniería de software	16
2.2. Imagen que ilustra el modelo Vista-Controlador	22
2.3. Imagen que ilustra un ejemplo de cómo opera el MVC	23
8.1. Ejemplo del encabezado de la matriz de pruebas	49
8.2. Ejemplo de la estructura de la matriz de pruebas	49
8.3. Informe de defectos Sprint 1 Ciclo 1	51
8.4. Gráfica de defectos por severidad Sprint 1 Ciclo 1	52
8.5. Gráfica de defectos por tipo de defecto Sprint 1 Ciclo 1	52
8.6. Gráfica de tipo de defectos por severidad Sprint 1 Ciclo 1	53

CAPÍTULO 1

Introducción

La etapa de mantenimiento de software requiere mayor tiempo y costo que sus fases complementarias, por lo que resulta ser la etapa de mayor complejidad dentro del ciclo de vida de desarrollo de software. Se estima que aproximadamente dos tercios del costo total del software se dedican al mantenimiento [1]. Esta situación es causada por diversos problemas presentes durante las etapas precedentes, principalmente en la etapa de análisis, ya que es difícil contar con las bases sólidas de una documentación bien construida y estructurada que favorezca a la fase de mantenimiento. Específicamente, el proceso de documentación de los casos de uso requiere una gran cantidad de esfuerzos humanos y es habitualmente propenso a errores, generando un impacto negativo en el desarrollo e implementación del sistema [2].

El desarrollo de una herramienta CASE (Ingeniería de Software Asistida por Computadora) favorecería la construcción y generación de la documentación de análisis ya que lograría estandarizar el estilo de trabajo que se emplea en la organización para documentar, elevaría la disponibilidad de la información de tal manera que los integrantes accedan a ella; controlaría quién escribe, modifica y supervisa cada parte del documento; finalmente, ayudaría en la generación de los documentos finales que se le entregan al cliente.

1.1. Problemática

La obtención de requerimientos es crucial para la generación de casos de uso desde el punto de vista del analista [3]. La inadecuada especificación de requerimientos es una de las causas predominantes en el fracaso del desarrollo de los sistemas de software hoy en día [4]. Del mismo modo, es común que el equipo de análisis se enfrente a situaciones que dificultan y prolongan la tarea de documentar casos de uso, algunos de los problemas más comunes son: [5]

- Falta de consistencia en la utilización de los nombres de actores, reglas de negocio y mensajes.
- Incorrecta agrupación de casos de uso en gestiones determinadas.
- Confusión entre escenarios.
- Falta de adaptación a un estándar de escritura y redacción de los elementos del documento.
- Incorrecta descripción de derechos funcionales (permisos).

Estas dificultades son resultado de la falta de experiencia de los analistas ya que el proceso de construcción del documento no es sencillo, al analista le toma tiempo aprender y hacer de manera entendible la redacción, la inclusión de elementos del caso de uso y la especificación correcta de las trayectorias. La curva de aprendizaje es extensa y es común que una persona inexperta en el tema tenga complicaciones y retrasos al realizar el documento, sin olvidar, el gran esfuerzo humano que requiere obtener un producto final óptimo. [6].

1.2. Propuesta

Se propone construir un sistema Web que asista en la generación de un documento de análisis basado en casos de uso que coadyuve a los analistas, de tal manera que puedan construir y generar sus documentos de forma estandarizada, que eleve la disponibilidad de la información contenida en los proyectos y que ayude al control del registro, edición y revisión de los casos de uso.

Para lo cual el sistema permitirá gestionar:

- Un catálogo de colaboradores, en donde el administrador podrá llevar un control de todo el personal involucrado en el proceso de desarrollo de los proyectos de la organización.
- Un catálogo de proyectos de administrador, en donde se podrá llevar un control del registro de los proyectos y el administrador tendrá la responsabilidad de capturar la información que corresponde a cada proyecto, tal como lo es: clave, nombre, fechas de término y fin, descripción, presupuesto y estado, así como asignar un líder por proyecto (seleccionado de los colaboradores previamente registrados). El administrador también tendrá la facultad de asignar un conjunto de colaboradores al proyecto, los cuales trabajarán como analistas y tendrán los permisos de acceso para operar en los proyectos que se le fueron encomendados.

- Un catálogo de términos de glosario en donde el colaborador controlará aquellas expresiones cruciales para el entendimiento de un proyecto en específico y podrá definirlos registrando su nombre y descripción, con el fin de tener consistencia en la utilización de los nombres de los términos.
- Un catálogo de actores el cual explicará brevemente el objetivo del mismo, teniendo la siguiente estructura para definirlos: el nombre del actor, descripción del mismo y sus responsabilidades relacionadas con el sistema según aplique, con el fin de tener consistencia en la utilización de los nombres de los actores.
- Un catálogo de Reglas de negocio especificando los siguiente: Identificador y nombre de la regla de negocio, de que tipo es, el nivel, una descripción explicando en qué consiste dicha regla, con el fin tener un control al momento de usarlas en diferentes casos de uso.
- Un catálogo de Mensajes el cual explicará brevemente el objetivo del mismo, este catálogo documentará los mensajes de la siguiente manera: identificador y nombre del mensaje, el tipo de mensaje, propósito, la redacción del mismo y que parámetros deben cumplirse para que el mensaje aparezca esto ayudará a que el usuario pueda reutilizar mensajes en diferentes casos de uso evitando la confusión de los nombres de los mensajes.
- Un catálogo de entidades con sus respectivos atributos.
- Un catálogo de pantallas con sus respectivas acciones.
- La agrupación de casos uso dividiéndolos por módulos. Con el propósito de obtener una estructura modular con alta cohesión, segmentando el conjunto de Casos de uso en partes mas pequeñas y con objetivos similares entre sí para que los analistas operen sobre una configuración más ordenada.
- La generación de los casos de uso integrando los elementos que forman y son parte del caso de uso.
- Un estándar de redacción y escritura definido para evitar confusiones en la descripción de los casos de uso.

Generar de manera semiautomatizada documentos de casos de uso a través de una herramienta Middle Case es un desafío que propone la idea de transformar la escritura del lenguaje natural (comúnmente empleado en la elaboración de dichos documentos) a un lenguaje estándar, formal y específico. De concretarse este desafío, el tiempo que actualmente toma solucionar los problemas que se presentan durante la elaboración del análisis y su documentación por el personal de análisis será optimizado. Coadyuvará a adquirir experiencia al equipo para disminuir errores, su uso representará una reducción en los recursos destinados al análisis y de este modo generar un documento de análisis con mayor estructura y consistencia.

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar un sistema que a través de una plataforma web asista en la generación de la documentación de casos de uso de un proyecto de software, a fin de contribuir en el proceso de su creación de tal manera que los integrantes del proyecto puedan documentar con base en un estándar, sobre una plantilla predefinida y características específicas.

1.3.2. Objetivos Específicos

1. Analizar la estructura de un documento de análisis basado en casos de uso para proponer una plantilla a construir.
2. Definir una arquitectura de trabajo a fin de que el desarrollo se base en esta.
3. Definir el alcance de los sprints con base en la metodología.
4. Implementar los módulos analizados y definidos en los sprints
5. Diseñar pruebas estáticas y dinámicas

Alcance

A continuación se explican los módulos que deberán satisfacer el sistema.

1. Generar un módulo de gestión de usuarios encargado del control de acceso y la administración de los usuarios, en este módulo se asignan los roles y permisos a los diferentes usuarios que se registren.
2. Generar un módulo de gestión proyectos encargado de la administración de proyectos, a partir de este módulo se hará el registro, lectura, modificación, eliminación de los componentes necesarios para la documentación del proyecto; contendrá un apartado para la asignación de usuarios al proyecto.
3. Generar un módulo de gestión de casos de uso encargado de la creación, lectura, modificación y eliminación de casos de uso así como la asociación de analistas.
4. Generar un módulo de gestión de elementos encargado de la definición y registro de todos los elementos necesarios para la creación de casos de uso, con el objeto de registrarlos en una base de datos y poder reutilizarlos al momento de escribir un caso de uso.
5. Generar un módulo de revisión y validación de casos de uso, encargado de mostrar los elementos que los conforman para su revisión y validación por usuarios permitidos.

6. Generar un módulo de generación de documento de análisis encargado de la generación del documento final de casos de uso para el desarrollo de un sistema con base en la plantilla predefinida.

1.4. Justificación

Un proyecto de software bien construido y formado es esencial para la competitividad de una organización dedicada al desarrollo de sistemas, e incluso para su propia supervivencia [7], del mismo modo, la documentación es un elemento partícipe que determina la calidad del sistema dado que [8]:

- Facilita la interpretación y comprensión del sistema.
- Provee los antecedentes que permiten conocer cómo fue diseñado, que hace y cómo está operando.
- Sirve de base para auditorias.
- Elimina los riesgos de dependencia con respecto al personal.
- Es fundamental para la capacitación de los usuarios del sistema facilitando la comunicación.
- Provee antecedentes esenciales, concretos y permanentes para evaluar modificaciones a su funcionamiento.
- Aumenta la seguridad y eficiencia en su mantenimiento reduciendo su costo.

Una herramienta web capaz de recolectar, almacenar y procesar los elementos que integran un proyecto para generar el documento de análisis será de gran apoyo para obtener un documento de calidad que logre satisfacer los puntos antes mencionados, de igual manera ayudará a los analistas, reduciendo de manera considerable el tiempo, costo y gastos de dicho documento.

El motivo por el cual se realizará este sistema radica en la necesidad e importancia de obtener un documento de análisis bien construido, es decir, a nivel análisis y a nivel herramienta:

- Favorecer la mantenibilidad del sistema en contrucción (a corto y largo plazo).
- Lograr una trazabilidad en los elementos del documento de casos de uso.
- Elevar la integridad y consistencia de la información del documento de análisis mediante los permisos que otorga el sistema.
- Incrementar la disponibilidad, con el documento cualquier persona con los permisos correspondientes va a poder realizar las tareas o acciones correspondientes con base en sus funciones.
- Conseguir un estándar en la forma de escribir el documento.

Y de esta manera, no solo resolver los problemas identificados en el proceso de construcción y generación del documento, si no también obtener una mejor calidad en dichos documentos que genera análisis, mismos que utiliza el resto del equipo en diferentes etapas del desarrollo y que se le entregan al cliente.

Éste proyecto se considera un trabajo terminal porque coadyuvará a formación de los autores en áreas de investigación, autoaprendizaje, y resolución de problemas, en la generación de este sistema se utilizarán conocimientos del área de Ingeniería de software, bases de datos, programación, tecnologías web, algoritmos y diseño orientado a objetos.

1.5. Estructura del Documento

El presente documento, está dirigido a todas aquellas personas interesadas en conocer el contenido del Trabajo Terminal 2018-B140, retoma los objetivos descritos en el protocolo, considerando las observaciones realizadas en la primera evaluación del trabajo. Se entregan otros dos documentos anexos a este reporte para un mejor entendimiento del sistema de los que al final de este capítulo se dará una breve explicación.

En el capítulo 2 Se expone el soporte conceptual de las definiciones teóricas que se utilizaron para el planteamiento del problema del trabajo terminal.

En el capítulo 3 Se muestra la situación actual en proyectos que tienen cierta relación con el trabajo terminal, en este análisis se muestran los avances más importantes que se han logrado con respecto al conocimiento de los generadores de casos de uso.

En el capítulo 4 Se expone el mercado al cual está enfocado el desarrollo de nuestro producto, así como la viabilidad de colocarlo en la industria en México.

En el capítulo 5 Se realiza la estimación de tiempo y costo con base en el método de puntos de función, técnica a través de la cual obtenemos un costo aproximado de la realización del proyecto.

En el capítulo 6 Se explica la arquitectura implementada en el sistema, así como las razones por las cuales se eligieron ciertas tecnologías.

En el capítulo 7 Se expone el avance y resultados obtenidos en los diferentes sprints de scrum que se estructuraron para el desarrollo del proyecto.

En el capítulo 8 Se detalla cual fue la metodología para la ejecución de pruebas, así como las técnicas para su evaluación, también se reportan los resultados de las pruebas ejecutadas.

Todo proyecto de software se desencadena por alguna necesidad de negocios: la de corregir un defecto en una aplicación existente, la de adaptar un “sistema heredado” a un ambiente de negocios cambiante, la de ampliar las funciones y características de una aplicación ya existente o la necesidad de crear un producto, servicio o sistema nuevo.

2.1. Ingeniería de software

Una de las primeras definiciones de ingeniería de software fue dada por Fritz Bauer en el año de 1969, quien define que la ingeniería de software es..

“el establecimiento y uso de principios robustos, orientados a obtener software económico que sea fiable y que funcione de manera eficiente sobre máquinas reales” . [9]

Aunque esta definición omite algunos términos referentes a tiempos de entrega, procesos eficaces, y calidad de software, nos da un panorama de sus principios fundamentales y es también la base de la definición que la IEEE ha desarrollado de una manera más completa:

“La ingeniería de software es: La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software.” [9]

La ingeniería de software está formada por un proceso, un conjunto de métodos (prácticas) y un arreglo de herramientas que permite a los profesionales elaborar software de cómputo de alta calidad.

Un aspecto muy importante de Ingeniería de Software es que proporciona parámetros formales para lo que se conoce como Gestión (o Administración) de Proyectos de Software. Esto se refiere a que Ingeniería de Software proporciona diversas métricas y metodologías que pueden usarse como especificaciones para todo lo referente a la administración del personal involucrado en proyectos de software, ciclos de vida de un proyecto de software, costos de un proyecto, y en si todo el aspecto administrativo que implica el desarrollar software.

La ingeniería en general es el análisis, diseño, construcción, verificación y gestión de entidades técnicas. En general, todo proceso de ingeniería debe comenzar por contestar las siguientes preguntas: ¿Cuál es el problema a resolver?, ¿Cuáles son las características de la entidad que se utiliza para resolver el problema?, ¿Cómo se realizará la entidad (y la solución)?, ¿Cómo se construirá la entidad?, ¿Cómo va a probarse la entidad?, y ¿Cómo se apoyará la entidad cuando los usuarios finales soliciten correcciones y adaptaciones a la entidad?. [9]



Figura 2.1: Capas de la ingeniería de software

La ingeniería de software es una tecnología con varias capas, como se muestra en la figura 2.1, existen 4 capas: herramientas, métodos, procesos y compromiso con la calidad. Cada una de ellas es importante, sin embargo, la capa de proceso es fundamental para el desarrollo de software, ya que es donde se define la estructura básica del producto hasta la culminación del mismo.

El proceso de software forma la base para el control de la administración de proyectos de software, y establece el contexto en el que se aplican métodos técnicos, se generan productos del trabajo (modelos, documentos, datos, reportes, formatos, etc.), se establecen puntos de referencia, se asegura la calidad y se administra el cambio de manera apropiada.[9]

2.2. Proceso de desarrollo de software

Se define proceso del software como una estructura para las actividades, acciones y tareas que se requieren a fin de construir software de alta calidad. La ingeniería de software es llevada a cabo por personas creativas y preparadas que deben adaptar un proceso maduro de software a fin de que resulte apropiado para los productos que construyen y para las demandas de su mercado. Cuando se trabaja en la construcción de un producto o sistema, es importante ejecutar una serie de pasos predecibles, una estructura general para la ingeniería de software se define en cinco actividades elementales: [9]

1. Comunicación
2. Planeación
3. Modelado
4. Construcción
5. Despliegue

Existen diferentes metodologías de desarrollo con modificaciones y adecuaciones al esquema general de construcción antes mencionado, algunas de ellas son las metodologías tradicionales y ágiles. Este proceso puede tener diferentes variaciones, sin embargo, sea cual sea la metodología aplicada, las etapas de Modelado (Análisis y Diseño) y Construcción (Codificación y Pruebas) son las más críticas e importantes para un producto final exitoso.

Durante el desarrollo, se realizan tareas específicas para cada etapa, por ejemplo, para la etapa de modelado se elabora el documento de análisis (donde se describe el funcionamiento del sistema), así como el diseño (en donde se genrean los diagramas que describen el funcionamiento establecido en el análisis); en la fase de construcción se genera el código del software y en la etapa de pruebas se valida y verifica que el software cumpla con lo asentado en las fases precedentes.

2.2.1. Análisis

El proceso de análisis dentro del desarrollo de software consiste en obtener los requerimientos del sistema para crear una solución, identificar los problemas a resolver o necesidad a ser atendida, evaluar las restricciones que presenta, así como los insumos se requieren para su debida construcción. Al ser la primera etapa dentro del proceso de desarrollo es la más crítica y sensible, ya que cualquier error de gran impacto que surja dentro de esta perjudicará las etapas consecuentes ocasionando retrasos en el proceso.[9]

En esta etapa se construye el documento de análisis, en donde se describen todos los requerimientos que el cliente ha solicitado mediante diferentes componentes. Con base en los requisitos, el analista sigue el proceso de análisis identificando en alto nivel que funcionalidades deberá poseer el sistema para cumplir los requerimientos. Una solución común para mapear cada funcionalidad es a través de CASOS DE USO.

Este documento emplea un lenguaje técnico especializado ya que busca ser comprendido por los diseñadores y programadores para su correcta construcción. [9]

Documento de análisis

Parte 1. Modelo de negocio:

- Glosario de términos.
- Modelo de información utilizado para representar la información que será almacenada en el sistema.
- Reglas de negocio mediante las cuales se normará el funcionamiento del sistema.

Parte 2. Modelo dinámico, el cual describe funcionalidad a partir de los siguientes capítulos:

- Arquitectura lógica.
- Máquinas de estados que modelarán el comportamiento de las entidades que así lo necesiten.
- Funciones y roles que tendrán los actores que interactuarán con el sistema.
- Casos de uso que describen funcionalidad.

Parte 3. Interacción con el usuario, que muestra las interfaces y mensajes a partir de los siguientes capítulos:

- Interfaces del sistema.
- Catálogo de mensajes.

Caso de Uso

Un caso de uso es una actividad que puede realizar un usuario dentro del software. Estas actividades sirven para describir el comportamiento del producto en distintas condiciones en las que el sistema responde a alguna de las peticiones realizadas por el usuario, es decir, describe el funcionamiento de los componentes acorde a las acciones que los usuarios realizan dentro del software.

En un libro que analiza cómo escribir casos de uso eficaces, Alistair Cockburn [Coc01b] afirma que “un caso de uso capta un contrato [...] [que] describe el comportamiento del sistema en distintas condiciones en las que el sistema responde a una petición de alguno de sus participantes[...]”.

En esencia, un caso de uso narra una historia estilizada sobre cómo interactúa un usuario final (que tiene cierto número de roles posibles) con el sistema en circunstancias específicas. La historia puede ser un texto narrativo, un lineamiento de tareas o interacciones, una descripción basada en un formato o una representación diagramática. Sin importar su forma, un caso de uso ilustra el software o sistema desde el punto de vista del usuario final.

Un caso de uso está compuesto por distintos elementos, los cuales se describen a continuación:

- **Actor:** Es la idealización de un rol que puede jugar una persona, otro sistema, proceso, un dispositivo o de alguna cosa que interactúa con el sistema. Los actores son objetos que residen fuera del sistema, en tanto que los casos de uso están compuestos por objetos y acciones que residen dentro del sistema. Todo actor tiene uno o más objetivos cuando utiliza el sistema. [12].
- **Entidad :** Representación de un objeto exclusivo único en el mundo real que se está controlando. Algunos ejemplos de entidad son una sola persona, un solo producto o una sola organización.
- **Atributo :** Es una especificación que define una propiedad de un objeto, elemento o archivo. También puede referirse o establecer el valor específico para una instancia determinada de los mismos.
- **Entrada:** Es la información producida por el usuario para ser guardada o procesada en el sistema. El usuario comunica y determina qué clases de entrada aceptará el sistema (por ejemplo, secuencias de control o de texto escritas a máquina a través del teclado y el ratón).
- **Salida:** Es la información producida por el sistema y percibida por el usuario. Las clases de salida los productos de programa, y las clases de entrada la que el programa acepta, definen la interfaz de usuario del programa.
- **Acción:** Evento originado por el usuario mediante botones.
- **Pantalla:** Es la interfaz de usuario, utiliza imágenes, iconos y menús para mostrar las acciones disponibles entre las que el usuario puede escoger en un sistema. Su función es proporcionar un entorno visual amigable y sencillo de usar que facilite la comunicación del usuario con el software.
- **Regla de Negocio:** Es aquella que rige los procesos de un negocio para garantizar el correcto funcionamiento del software. Las reglas de negocio establecen los procedimientos que se deben realizar y las condiciones sobre las que dichas actividades se van a ejecutar.
- **Mensaje:** Constituyen la mínima unidad de comunicación entre el usuario y el sistema. Se trata de un proceso de comunicación completa porque el sistema lanza un mensaje hacia el usuario que no se resuelve hasta que el usuario lo recibe o lo responde, completando así el proceso de comunicación con la realimentación correspondiente.
- **Trayectoria:** Es un conjunto de pasos que describen la interacción entre el usuario y el sistema.
- **Paso:** Es una instrucción que realiza el usuario o el sistema.
- **Precondición:** Está formada por el conjunto de condiciones que se tienen que cumplir para que se pueda iniciar un caso de uso. En muchos casos supone la ejecución de casos de uso previos.
- **Postcondición:** Refleja el estado en que se queda el sistema una vez ejecutado el caso de uso.
- **Puntos de extensión:** Es la incorporación implícita del comportamiento de otro caso de uso, el cuál no es parte del flujo principal. Modela la parte opcional del sistema, un subflujo que sólo se ejecuta bajo ciertas condiciones o varios flujos que se pueden insertar en un punto determinado.

2.3. Herramienta CASE

2.3.1. Definición

Las tecnologías de ingeniería de software asistida por computadora (CASE) son herramientas que brindan asistencia automatizada para el desarrollo de software. El objetivo de presentar las herramientas CASE es la reducción del tiempo y el costo del desarrollo de software y la mejora de la calidad de los sistemas desarrollados. [15]

El interés en las herramientas y entornos de CASE se basa en las expectativas de:

- Aumentar la productividad
- Mejorar la calidad del producto
- Facilitar el mantenimiento
- Hacer que la tarea de los ingenieros de software sea menos odiosa y más agradable.

Una herramienta CASE se puede clasificar en tres categorías :

- Upper CASE (U-CASE), herramientas que ayudan en las fases de planificación, análisis de requisitos y estrategia del desarrollo, usando, entre otros diagramas UML.
- Middle CASE (M-CASE), herramientas para automatizar tareas en el análisis y diseño de la aplicación.
- Lower CASE (L-CASE), herramientas que semi-automatizan la generación de código, crean programas de detección de errores, soportan la depuración de programas y pruebas. Además automatizan la documentación completa de la aplicación. Aquí pueden incluirse las herramientas de desarrollo rápido de aplicaciones.

Debido a sus similitudes, a veces Upper CASE y Middle CASE simplemente se conocen como Upper CASE. En general, Upper CASE es una herramienta para una vista de alto nivel del desarrollo de software, mientras que lower CASE se utiliza principalmente como herramienta en la fase de programación y prueba. [16]

2.4. Implementación del desarrollo

2.4.1. Patrones de diseño

Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, también describe el núcleo de la solución a ese problema, de tal manera que puede usar esta solución un millón de veces, sin tener que hacerlo lo mismo dos veces. [19]

En general, un patrón tiene cuatro elementos esenciales [17]:

1. El nombre del patrón es un identificador que podemos usar para describir un problema de diseño, sus soluciones y consecuencias en una o dos palabras. Nombrar un patrón nos permite diseñar a un nivel más alto de abstracción.
2. El problema describe cuándo aplicar el patrón. Explica el problema y su contexto. Podría describir problemas de diseño específicos, como la forma de representar algoritmos como objetos. Podría describir estructuras de clase u objeto que son sintomáticas de un diseño inflexible. A veces, el problema incluirá una lista de condiciones que deben cumplirse antes de que tenga sentido aplicar el patrón.
3. La solución describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño o implementación concreta en particular, porque un patrón es como una plantilla que se puede aplicar en muchas situaciones diferentes. En cambio, el patrón proporciona una descripción abstracta de un problema de diseño y cómo lo resuelve una disposición general de elementos (clases y objetos en nuestro caso).
4. Las consecuencias son los resultados y las compensaciones de aplicar el patrón. Aunque las consecuencias a menudo no se expresan cuando describimos las decisiones de diseño, son críticas para evaluar las alternativas de diseño y para comprender los costos y beneficios de aplicar el patrón. Las consecuencias para el software a menudo se refieren a compensaciones de espacio y tiempo. También pueden abordar problemas de lenguaje e implementación. Dado que la reutilización es a menudo un factor en el diseño orientado a objetos, las consecuencias de un patrón incluyen su impacto en la flexibilidad de un sistema, extensibilidad o portabilidad. Enumerar estas consecuencias explícitamente le ayuda a comprenderlas y evaluarlas.

Patrón de Diseño MVC (Modelo Vista-Controlador)

El patrón de diseño de modelo-vista-controlador (MVC) especifica que una aplicación consta de un modelo de datos, de información de presentación y de información de control. El patrón requiere que cada uno de estos elementos esté separado en distintos objetos. [17]

Características

MVC consta de tres tipos de objetos:

1. El modelo es el objeto de la aplicación (por ejemplo, la información de datos) contiene únicamente los datos puros de aplicación; no contiene lógica que describe cómo pueden presentarse los datos a un usuario.
2. La vista es su presentación en pantalla (por ejemplo, la información de presentación) presenta al usuario los datos del modelo. La vista sabe cómo acceder a los datos del modelo, pero no sabe el significado de estos datos ni lo que el usuario puede hacer para manipularlos.

3. Por último, el controlador que define la forma en que la interfaz de usuario reacciona a la entrada del usuario (por ejemplo, la información de control) está entre la vista y el modelo. Escucha los sucesos desencadenados por la vista (u otro origen externo) y ejecuta la reacción apropiada a estos sucesos. En la mayoría de los casos, la reacción es llamar a un método del modelo. Puesto que la vista y el modelo están conectados a través de un mecanismo de notificación, el resultado de esta acción se reflejará automáticamente en la vista.

El comportamiento se ilustra en la figura 2.2

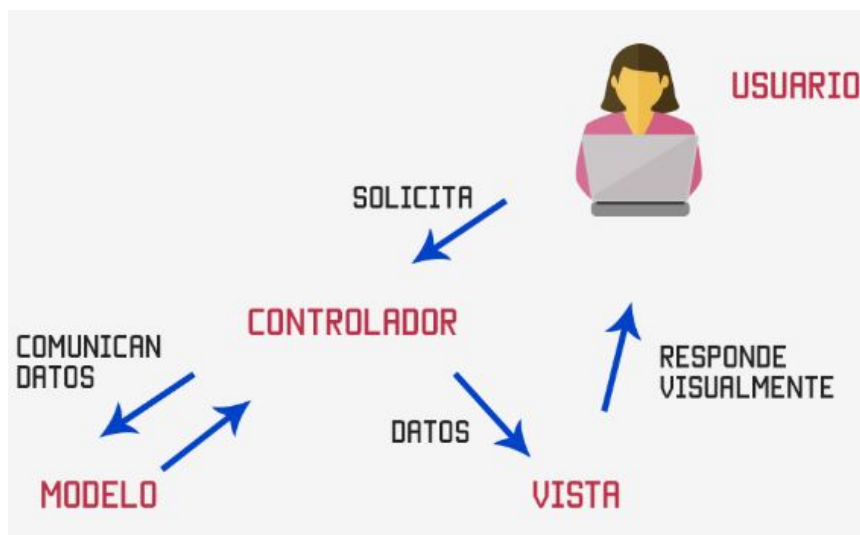


Figura 2.2: Imagen que ilustra el modelo Vista-Controlador

MVC desacopla las vistas y los modelos al establecer un protocolo de suscripción / notificación entre ellos. Una vista debe garantizar que su apariencia refleje el estado del modelo. Cada vez que los datos del modelo cambian, el modelo notifica las vistas que dependen de él. En respuesta, cada vista tiene la oportunidad de actualizarse. Este enfoque le permite adjuntar múltiples vistas a un modelo para proporcionar diferentes presentaciones. También puede crear nuevas vistas para un modelo sin reescribirlo.

El siguiente diagrama muestra un modelo y tres vistas. (Se omitieron los controladores por simplicidad). El modelo contiene algunos valores de datos, y las vistas que definen una hoja de cálculo, un histograma y un gráfico circular muestran estos datos de varias maneras. El modelo se comunica con sus vistas cuando cambian sus valores, y las vistas se comunican con el modelo para acceder a estos valores. [20]

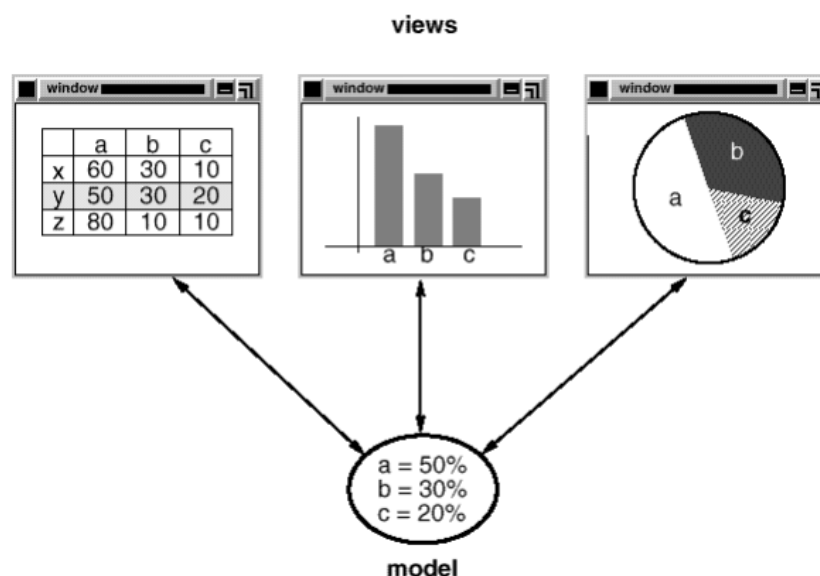


Figura 2.3: Imagen que ilustra un ejemplo de cómo opera el MVC

2.4.2. Framework

Definición

Es un conjunto de clases que encarna un diseño abstracto para soluciones a una familia relacionada de problemas. [17].

Conjunto de clases cooperativas que construyen un diseño reutilizable para un tipo específico de software. Un Framework proporciona la arquitectura partiendo el diseño en clases abstractas y definiendo sus responsabilidades y colaboraciones. Un desarrollador realiza una aplicación haciendo subclases y componiendo instancias a partir de las clases definidas por el Framework.

Un framework es una aplicación semi-completa, de control invertido, así un Framework difiere de una librería de clases [17]:

- En una librería de clases, el control del flujo se encuentra en el código de la aplicación que realiza llamadas a los métodos de la librería de clases.
- En un framework, el control del flujo está en código del framework que realiza llamadas al código de la aplicación (control invertido).

Ventajas de un framework

Como ventajas en la utilización de un framework tenemos las siguientes:

1. Minimiza tiempos de desarrollo / acorta el “Time to market”

- Los proyectos de desarrollo ya no tendrán que resolver los múltiples problemas asociados a las aplicaciones web.
- Los frameworks reducen la codificación y sobretodo la puesta en marcha, ya que proporcionan subsistemas que sabemos que ya funcionan.
- En definitiva, proporcionan código que no se tendrá que mantener ni reescribir.

2. Reduce los riesgos del desarrollo.

- Con un modelo de programación complejo como el de J2EE, el riesgo de fallos en los proyectos iniciales es alto.

3. Proporciona una arquitectura consistente entre aplicaciones.

- Al usar frameworks todas las aplicaciones generadas comparten una arquitectura común. Esto hace que sea más fácil de aprender, mantener y soportar.
- Cualquier programador que trabaje con un framework no deberá invertir gran parte de su tiempo en buscar las clases necesarias, interconectarlas o descubrir los métodos que contienen. Los frameworks ocultan toda esta complejidad dando un alto nivel de abstracción.

2.5. Pruebas

Con base en los enunciados de la ISTQB (International Software Testing Qualifications Board). [34] Las pruebas de software muestran la presencia de defectos: todo tipo de software que se desarrolle es susceptible a la presencia de «Bug's» o defectos, con el testing de software se busca reducir al máximo la presencia de estos, pero no se puede asegurar que el software esté libre de defectos, al contrario, con el tiempo las modificaciones o adiciones de funcionalidades aumenta el riesgo de la presencia de nuevos defectos.

Las pruebas dinámicas y estáticas son métodos complementarios, ya que tienden para encontrar diferentes tipos de defectos de manera efectiva y eficiente. Tipos de defectos que son más fáciles de encontrar durante las pruebas estáticas son: desviaciones de los estándares, falta requisitos, defectos de diseño, código no mantenible e interfaz inconsistente especificaciones. Tenga en cuenta que, a diferencia de las pruebas dinámicas, las pruebas estáticas encuentran defectos en lugar de fallas.

2.5.1. Pruebas Dinámicas

La ISTQB (International Software Testing Qualifications Board) dice que la pruebas dinámicas son todas aquellas pruebas que para su realización requieren la ejecución de la aplicación. Con los métodos de prueba dinámicos, el software se ejecuta utilizando un conjunto de valores de entrada y su salida se examina y compara con lo que se espera. Las pruebas dinámicas permiten el uso de técnicas de caja negra

y caja blanca con mayor amplitud.

Dentro de las pruebas dinámicas podemos encontrar distintos niveles de prueba. Los niveles de prueba son grupos de actividades de prueba que se organizan y gestionan juntas. Cada nivel de prueba es una instancia del proceso de prueba, realizadas en relación con software en un determinado nivel de desarrollo, desde unidades individuales o componentes hasta sistemas completos o, en su caso, sistemas de sistemas.

Los niveles de prueba utilizados pueden ser:

- Prueba de componentes.
- Pruebas de integración.
- Prueba de sistema.
- Pruebas de aceptación.

Los niveles de prueba se caracterizan por los siguientes atributos:

- Objetivos específicos
- Base de prueba, referenciada para derivar casos de prueba
- Objeto de prueba (es decir, lo que se está probando)
- defectos y fallas típicas

2.5.2. Prueba de sistema

Objetivos de las pruebas del sistema.

Las pruebas del sistema se centran en el comportamiento y las capacidades de un sistema o producto completo, a menudo considerando las tareas de extremo a extremo que el sistema puede realizar y los comportamientos no funcionales que exhibe mientras realiza esas tareas. Los objetivos de las pruebas del sistema incluyen:

- Reducción del riesgo.
- Verificar si los comportamientos funcionales y no funcionales del sistema son los diseñados y especificado
- Validar que el sistema está completo y funcionará como se esperaba.
- Generar confianza en la calidad del sistema en su conjunto.

- Encontrar defectos
- Evitar que los defectos escapen a niveles de prueba o producción más altos

Base de prueba

Los ejemplos de productos de trabajo que se pueden usar como base de prueba para las pruebas del sistema incluyen:

- Especificaciones de requisitos del sistema y software (funcionales y no funcionales)
- Informes de análisis de riesgos.
- Casos de uso
- Epics e historias de usuarios
- Modelos de comportamiento del sistema.
- Diagramas de estado
- Sistema y manuales de usuario.

Objetos de prueba

Los objetos de prueba típicos para las pruebas del sistema incluyen:

- Aplicaciones
- Sistemas de hardware / software
- Sistemas operativos
- Sistema bajo prueba (SUT)

2.5.3. Pruebas Estáticas

Durante las pruebas estáticas, los productos de trabajo de software se examinan manualmente o con un conjunto de herramientas, pero no se ejecutan. Como consecuencia, las pruebas dinámicas solo pueden aplicarse al código de software. La ejecución dinámica se aplica como una técnica para detectar defectos y para determinar los atributos de calidad del código.

Es común que dentro del área de la ingeniería de software, se confundan los términos: "Caso de uso" y "Diagrama de caso de uso", sin embargo es importante resaltar las diferencias para comprender el objetivo principal del proyecto terminal.

Un caso de uso narra una historia detallada sobre cómo interactúa un usuario final (con cierto número de roles posibles) con el sistema en circunstancias específicas. La historia puede ser un texto narrativo, un lineamiento de tareas o interacciones, una descripción basada en un formato o una representación diagramática de casos de uso. Sin importar su forma, un caso de uso ilustra el software o sistema desde el punto de vista del usuario final [9].

En otras palabras, un caso de uso es aquel que describe en forma de secuencia de acciones o pasos la interacción entre un actor y el sistema, en cambio, un diagrama de casos de uso es una representación visual simple de las interacciones del sistema con el mundo exterior, el modelo de un grafo con dos tipos de nodos (Actor y caso de uso), el cual ilustra gráficamente el comportamiento del caso de uso. Un diagrama de casos de uso no describe la interacción detallada del sistema con los actores ni reemplaza o sustituye el concepto de caso de uso.

Ahora bien, en la red hay una gran variedad de sistemas que permiten la generación de **diagramas de casos de uso en UML**, a partir de distintas técnicas, sin embargo no hay herramientas comerciales o gratuitas que posibiliten la generación del documento con las especificaciones y la gestión de sus componentes, tal como lo pretende el presente trabajo terminal.

3.1. Antecedentes

3.1.1. UCD-Generator - Una aplicación LESSA para el diseño de casos de uso

Las herramientas CASE convencionales requieren una comprensión completa del negocio, una gran cantidad de tiempo y esfuerzos adicionales por parte del analista del sistema durante el proceso de creación, organización, etiquetado y finalización de los diagramas de casos de uso. Es por esto que se diseñó un sistema que proporciona una manera rápida y confiable de generar diagramas de casos de uso para ahorrar tiempo y presupuesto tanto para el usuario como para el analista del sistema.

Objetivo

Este sistema presenta un enfoque basado en el procesamiento del lenguaje natural LESSA (Language Engineering System for semantic analysis) que se utiliza para comprender automáticamente el texto en lenguaje natural y extraer la información requerida. Esta información se utiliza para dibujar los diagramas de casos de uso. El usuario escribe sus preferencias basadas en la interfaz en inglés, en unos pocos párrafos y el sistema diseñado tiene una capacidad notable para analizar el script dado. Después del análisis compuesto y la extracción de información asociada, el sistema diseñado en realidad dibuja los diagramas de casos de uso [10].

3.1.2. Generación automatizada de diagramas de casos de uso a partir de requerimientos de usuarios

Con el estado actual de la tecnología de procesamiento de lenguaje natural (PNL), muchos investigadores han demostrado que automatizar el proceso de análisis de requisitos es posible, lo que ahorra una cantidad significativa de tiempo invertido por los analistas. Se han desarrollado numerosas herramientas semiautomáticas que ayudan al analista en este proceso. Sin embargo, una técnica comúnmente utilizada para usar la gramática en el texto obtenido como la base para identificar información útil, ha estado enfrentando problemas de escalabilidad debido a que el formato textual de los requisitos consiste en lenguaje natural no estructurado (NL).

Objetivo

Este proyecto utiliza una técnica probabilística para identificar actores y casos de uso. El resultado prometedor demuestra que las mejoras adicionales de este enfoque pueden automatizar completamente la fase de análisis, propone una metodología para la asistencia automática de análisis de requisitos a los analistas de software mediante la extracción de un diagrama de caso de uso del documento de requisitos del usuario. Este proyecto ha intentado con éxito extraer actores y usar casos utilizando un modelo de clasificación probabilística junto con una asistencia mínima de enfoque basado en reglas. Los casos de uso son nítidos y consistentes independientemente del tamaño del texto de los requisitos. Debido al pequeño tamaño de los datos utilizados, el rendimiento no se ha logrado precisar. Sin embargo, se pueden utilizar

mejores modelos de clasificación con un conjunto de datos más grande que incluya otros dominios de software para mejorar los resultados. El desafío restante aquí se relaciona con abordar los requisitos no funcionales y también para incorporar funciones de inclusión y extensión al diagrama de casos de uso. Un gráfico bien diseñado [\[12\]](#).

Análisis de mercado

En este apartado se demuestra la viabilidad comercial del trabajo terminal TESSERACT en México, así mismo se realiza un estudio en donde se determina el campo en donde un sistema con las características del generador de documento de casos de uso podría generar un mayor impacto y aceptación por parte de los equipos de desarrollo de software. Cabe resaltar que TESSERACT no pretende ser comercializado por el momento.

4.1. Situación actual y evolución del mercado

El software es un elemento consustancial a la economía moderna, es uno de los sectores tecnológicos más competitivos, se usa en una gran cantidad de productos manufacturados y servicios, por lo que la elaboración de programas de cómputo figura en casi todas las industrias y es, de hecho, factor de éxito de todos los sectores de la economía. Esta industria ha tenido una evolución constante en lo que se refiere a las metodologías o bien, las formas en las cuales se realiza la planeación para el diseño del software, básicamente con el objetivo de mejorar, optimizar procesos y ofrecer una mejor calidad. [22].

En el campo del desarrollo de software, existen dos grupos de metodologías, las denominadas tradicionales (formales) y las ágiles. Las primeras son un tanto rígidas, exigen una documentación exhaustiva y se centran en cumplir con el plan del proyecto definido totalmente en la fase inicial del desarrollo del mismo; mientras que la segunda enfatiza el esfuerzo en la capacidad de respuesta a los cambios, las habilidades del equipo y mantener una buena relación con el usuario. La metodología que sea seleccionada, debe ser adaptada al contexto del proyecto, teniendo en cuenta los recursos técnicos y humanos; tiempo de desarrollo y tipo de sistema. [24].

Dean Leffingwell, autor de *Scaling Software Agility*, menciona que los Casos de Uso son una herramienta valiosa para modelar requerimientos en metodologías Lean/Ágiles de gran envergadura, si embargo, no es común encontrar casos de uso en los proyectos ágiles (especialmente en XP y Scrum), en donde se suele utilizar historias de usuario para recolectar los requerimientos [12].

Ahora bien, de acuerdo a la teoría expuesta, TESSERACT al ser una herramienta que asiste a la generación del documento de casos de uso, se convierte en un instrumento que puede contribuir en cualquier metodología, ya sea formal o ágil, sin embargo el beneficio e impacto incrementa cuando se utiliza en la construcción de sistemas con metodologías formales y de gran escala, los casos de uso son una herramienta muy poderosa para explorar las interacciones entre los usuarios, los sistemas, y los sub-sistemas. Más aún, la técnica de casos de uso es la mejor forma para identificar todos los escenarios alternativos que se nos aparecen, fundamentales para asegurar la calidad de los sistemas. En los desarrollos ágiles, los casos de uso no reemplazan a las historias de usuario pero pueden resultar sumamente útiles para analizar, elaborar y comprender el funcionamiento deseado de sistemas complejos.

4.1.1. Industria Mexicana del Software

Para conocer el nivel de oportunidad que tiene TESSERACT dentro de la industria en México, es importante conocer de manera cuantitativa cual es el perfil de las empresas desarrolladoras de software en México. Es importante destacar que los diversos análisis que hasta la fecha se han realizado con respecto al panorama de este sector no resultan aún generalizables a toda la industria, ya que cada estudio analiza sólo un subconjunto del total de empresas, por lo tanto se hace la aclaración que lo aquí se presenta son datos representativos, y no necesariamente significa que sean generalizables.

Localización Geográfica de las Empresas Participantes

Las empresas participantes en el estudio se localizan en 11 de los 32 estados de la República Mexicana, presentando la siguiente distribución: 2.9 % Chihuahua, 1.5 % en Coahuila, 44.1 % en la Ciudad de México, 11.8 % en Durango, 2.9 % en el Estado de México, 1.5 % en Guanajuato, 2.9 % en Jalisco, 2.9 % en Michoacán, 2.9 % en Morelos, 23.5 % en Nuevo León y 2.9 % en Querétaro. Esta concentración es similar a la de otros estudios realizados para este sector en México.[13]

Número de Empresas Desarrolladoras de Software en México

La respuesta a esta pregunta no tiene una cifra exacta. De acuerdo con estimaciones realizadas por ESANE consultores sobre del número total de empleados y empresas de la Industria del Software en México, el número aproximado de empresas de la industria mexicana del software podría ser del orden de 1,500 empresas.[13].

Tamaño de las Empresas

El estudio revela que el 85.29 % de las empresas del sector de la Industria Mexicana del Software son de tamaño micro (54.41 %) y pequeño (30.88 %), el 5.8 % mediana, y tan sólo el 8.82 % son de tamaño grande (con un número de empleados mayor a 100) [13].

Las oportunidades que se tienen de posicionar TESSERACT en el mercado son amplias, la industria del software muestra un constante crecimiento y en México hay posibilidades de colocar nuestra herramienta en diversos sectores para contribuir en el desarrollo y construcción de software.

Estimación de tiempo y costo

5.1. Puntos de función

Es una técnica de estimación de software desarrollada originalmente por Allan Albrecht en 1979 mientras trabajaba para IBM, quien definió conceptos para medir el software a partir de valoraciones de funcionalidades entregadas al usuario y no a partir de aspectos técnicos, con la intención de producir valoraciones independientes de la tecnología y fases del ciclo de vida utilizado.

El trabajo de Albrecht fue continuado por el grupo internacional de usuarios de puntos de función, quienes plasmaron sus conceptos en el método IFPUG-FPA (International Function Points Users Groups), el cual realiza las valoraciones a partir de la funcionalidad del sistema, primero clasificándolas, luego asignando una complejidad y ponderación a cada una según unas tablas predefinidas, determinando así el valor de puntos de función.

Sumando los puntos de todas las funcionalidades se obtiene la valoración de todo el proyecto y finalmente se puede aplicar un factor de ajuste, que puede depender de características generales del sistema como por ejemplo requerimientos no funcionales como el rendimiento, reusabilidad, facilidad de instalación y operación entre otros aspectos.

Los puntos de función permiten traducir el tamaño de funcionalidades de software a un número, a través de la suma ponderadas de las características que este tiene. Una vez que tenemos los puntos de función, podemos traducirlos en horas hombre o días de trabajo, según factor de conversión que dependería de mediciones históricas de nuestra productividad. Con las horas hombre, podemos determinar el costo y presupuesto de los proyectos. [12].

Desarrollaremos la medición en dos pasos, primero determinaremos los componentes funcionales del presupuesto de desarrollo de software, a partir del análisis de requerimientos realizado anteriormente. Seguidamente, realizaremos el cálculo de los puntos de función, con lo cual obtendremos una medida del tamaño del proyecto.

Para determinar los componentes funcionales, debemos determinar tanto las transacciones de negocio como los componentes de datos, siguiendo el método de análisis de puntos de función. Las transacciones de negocio que podemos desglosar a partir de los requerimientos de software son las siguientes.

Tipo de Caso de uso	Cantidad
Gestionar	13
Registrar	18
Modificar o Editar	15
Eliminar	17
Buscar	9

Seguidamente, clasificamos las transacciones de negocio, que pueden ser de 3 tipos: Entradas, salidas y consultas.

- Entradas: Registrar modificar y eliminar
- Salida: Gestionar
- Consultas: Buscar
- Archivo lógico interno: Tablas en Base de datos

Adicionalmente, debemos asignar un nivel de complejidad alto, medio o bajo a cada uno con base en la siguiente tabla.

Tipo	Baja	Media	Alta
Entrada externa (EI)	3PF	4PF	6PF
Salida externa (EO)	4PF	5PF	7PF
Consulta externa (EQ)	3PF	4PF	6PF
Archivo lógico interno (ILF)	7PF	10PF	15PF
Archivo de interfaz externo (ILF)	5PF	7PF	10PF

Los niveles de complejidad dependen de factores como por ejemplo el número de campos no repetidos, número de archivos a ser leídos, creados o actualizados, número de sub grupos de datos o formatos de registros, entre otros.

Al clasificar las transacciones de negocio y asignar los niveles de complejidad se llegó a la conclusión que para el desarrollo de TESSERACT, la complejidad en todos sus niveles es **MEDIA**

Al determinar los puntos de función tenemos una medida de la magnitud del tamaño del software y del esfuerzo que se requiere para desarrollarlo.

Tipo de Caso de uso	Cantidad	Complejidad	Total PF
Gestionar	13	5	65
Registrar	18	4	72
Modificar o Editar	15	4	60
Eliminar	17	4	68
Buscar	9	4	36
Tablas en BD	65	10	650

MAGNITUD ESTIMADA: 951 PF

Lenguaje	Horas medio	PF Pro-	Lineas de código por PF
Lenguajes de 4ta generación	8		20

HORAS HOMBRE: 7608 hrs para que una persona termine el sistema.

Ahora bien para estimar el tiempo se tienen los siguientes datos:

Concepto	Tiempo
Desarrolladores	4
Horas de trabajo al día	8
Días al mes de trabajo	24
Horas de trabajo x desarrollador	1902
Días de trabajo x desarrollador	237
Meses de trabajo	10

TIEMPO ESTIMADO: 10 meses para desarrollar el software, con un trabajo de lunes a sábado, 8 hrs diarias con 4 desarrolladores.

Para estimar el costo del proyecto se tiene la siguiente información:

TIEMPO ESTIMADO: 10 MESES

DESARROLLADORES: 4 DESARROLLADORES

Concepto	Cantidad	Total
Sueldo Mensual de un desarrollador	\$18,000	\$720,000
Consumo de luz por mes	\$125	\$1250
Consumo de agua por mes	\$50	\$500
Otros costos del proyecto por mes	\$850	\$8500

Datos salariales promedios obtenidos de <https://www.indeed.com.mx/salaries/Desarrollador/a-java-Salaries>

COSTO TOTAL ESTIMADO: \$730,250

La arquitectura implementada tiene como finalidad que el sistema web que se desarrolle sea estable, modular y escalable; para su realización se utilizaron las siguientes tecnologías:

6.1. Frameworks

6.1.1. Spring

Spring nos brinda ventajas como: la administración de objetos, garantizando que los recursos se creen, reutilicen y limpien correctamente (Managed Objects) [25]; El Manejo de transacciones declarativas, por ejemplo Rollback de una transacción [26] y un modelo de programación consistente en diferentes API de transacciones, tal como Java Persistence API. La utilización de Spring junto con sus características nos brinda un sistema con alta disponibilidad.

6.1.2. Struts

Se implementó Struts2 porque es un framework de código abierto, extensible mediante plugins (REST, AJAX, etc) y utiliza el patrón de diseño MVC. [27]

Las ventajas ofrecidas por este framework son:

- Patron Observador (Observer pattern), las vistas se actualizan por sí mismas del modelo. [28]
- Provee un bajo acoplamiento entre la vista y el modelo haciendo aplicaciones significativamente más fáciles de crear y mantener. [28]

- Uso de Interceptores, ejecutan código antes y después de que un Action es invocado, por ejemplo al subir un archivo [\[29\]](#)

6.2. Modelo MVC

El patrón de diseño MVC (Modelo Vista Controlador) es fundamental para la separación entre lógica de interfaz de usuario y lógica de negocio [\[30\]](#); de esta manera la estructura del proyecto tiene las siguientes características [\[31\]](#):

- Los objetos del modelo de datos encapsulan información.
- Los objetos de vista muestran información al usuario.
- Los objetos del controlador implementan acciones.
- Los objetos de vista observan los objetos del modelo de datos y actualizan su visualización cada vez que cambian los objetos del modelo.
- Los objetos de vista recopilan la entrada del usuario y la pasan a un objeto controlador que realiza la acción.

6.3. Arquitectura N Capas

La arquitectura n capas está implementada dentro del patrón MVC en el Modelo, ayuda a la modularidad de la capa de negocio, la capa de reglas de negocio y la capa de persistencia, de tal manera que se consigue un bajo acoplamiento.

6.4. Patrones de Diseño

6.4.1. Interface

El patrón interface se utilizó para denotar aquellas clases que pueden ser utilizadas por el DAO genérico y realizar las operaciones elementales a la base de datos

6.4.2. Singleton

El patrón de diseño singleton se implementó mediante anotaciones de Spring, cuando un bean es anotado con el alcance Singleton, solo se administrará una instancia compartida del bean, y todas las solicitudes de beans con un id o ids que coincidan con esa definición de bean harán que el contenedor

Spring devuelva una instancia de bean específica [28], de esta manera se convierte en un objeto administrado por Spring.

La ventaja de esta arquitectura es la capacidad de adaptación, si en algún momento se debe modificar alguna capa, por ejemplo la capa de persistencia, ésta modificación no afectaría a las otras capas ya que la comunicación entre ellas es transparente.

6.5. Logger log4j

CAPÍTULO 7

Metodología

En este capítulo se describen las actividades realizadas durante cada uno de los sprints planteados para la entrega del trabajo terminal.

De acuerdo a las necesidades del proyecto, se determinó trabajar con base en la metodología scrum empleando específicamente el uso de sprints.

El núcleo central de la metodología de trabajo ‘scrum’ es el ‘sprint’. Se trata de un miniproyecto de no más de un mes (ciclos de ejecución muy cortos -entre una y cuatro semanas), cuyo objetivo es conseguir un incremento de valor en el producto que estamos construyendo. Todo ‘sprint’ cuenta con una definición y una planificación que ayudará a lograr las metas marcadas. [21]

Scrum, que se emplea cada vez con más frecuencia para desarrollar productos y servicios digitales, es un marco de trabajo donde los miembros de un equipo colaboran en la construcción de un producto de manera que sea valioso desde sus primeras etapas. Para lograr esta entrega de valor tan rápida y continua, los equipos ‘scrum’ trabajan en ciclos de ejecución muy cortos -entre una y cuatro semanas- que se denominan ‘sprints’ y tienen un objetivo muy claro. [21]

El primer paso para alcanzar este objetivo -o hito del proyecto- es la reunión de planificación, una sesión en la que debe participar todo el equipo ‘scrum’ y que supone el pistoletazo de salida del ‘sprint’. Esta reunión se divide en dos partes que tratan de dar respuesta a dos preguntas fundamentales: ¿Qué se va a entregar? y ¿cómo se va a realizar el trabajo?, donde cada miembro inspecciona el trabajo de los otros para poder hacer las adaptaciones necesarias, comunica cuales son los conflictos en los que se encuentra, actualiza el estado de la lista de tareas de la iteración (Sprint Backlog) y los gráficos de trabajo pendiente (Burndown charts). [22]

7.1. Sprint 0: Configuración del ambiente de trabajo

7.1.1. Selección de herramientas

Elemento	Herramienta	Justificación
Sistema de composición de textos	Latex	Es un sistema de software libre y de composición tipográfica de alta calidad, esta herramienta incluye características diseñadas para la producción de la documentación técnica en donde se establecen las características del proyecto.
Lenguaje de programación	Java	La idea principal de TESSERACT es que opere como un sistema web, java al ser un lenguaje multiplataforma nos permite construir un producto con las características deseadas. Otras de las ventajas que tiene java son sus librerías estándar, la facilidad de programación y su paradigma orientado a objetos.
Framework	Struts	Struts permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en las que Java Enterprise esté disponible lo convierten en una herramienta altamente disponible.
SGBD	Mysql	MySQL es un sistema gestor de base de datos muy popular, es usado por su simplicidad y rendimiento. Es la opción para algunas aplicaciones comerciales gracias a su facilidad de uso y el poco tiempo que requiere para ponerse en marcha.

7.2. Sprint 1: Gestión de colaboradores

7.2.1. Iniciar sesión

Análisis

Se requiere un proceso mediante el cual se controle el acceso individual al sistema mediante la identificación de usuario utilizando credenciales provistas por el registro de personal.

7.2.2. Gestionar colaborador

El actor puede visualizar aquellas personas de la organización que se encuentran involucradas en uno o más proyectos, del mismo modo el actor puede realizar diferentes acciones, como un nuevo registro o consultar, modificar y eliminar personas previamente registradas.

Análisis

Es importante tener un mecanismo de control sobre aquellas entidades humanas que participan en las actividades, toma de decisiones y otras funciones dentro de un proyecto de software. Estas personas pueden estar involucradas en uno o más proyectos. El administrador, tiene la facultad y poder de:

Registrar colaborador: Para el registro fué de suma importancia establecer los datos de entrada más relevantes para el sistema, esto con el fin de no saturar la base de datos.

La información solicitada para el registro de personal es la siguiente:

- **CURP:** Con el fin de tener unicidad de registros en la base de datos.
- **Nombre, Primer Apellido, Segundo Apellido:** Con el fin de identificar a las personas.
- **Correo electrónico:** Para utilizarse como nombre de usuario en el inicio de sesión.
- **Contraseña:** Con el fin de tener una autenticación segura y comprobada en el sistema.

Una vez que se realiza el registro de una persona de manera exitosa, esta podrá participar en algún proyecto.

Modificar colaborador: Dentro del sistema existe la posibilidad de que los datos de una persona previamente registrada sean modificados, a excepción de la CURP.

Eliminar colaborador: El sistema debe permitir al administrador eliminar el registro de una persona siempre y cuando esta no lidere ningún proyecto.

7.3. Sprint 2: Gestión de proyectos

7.3.1. Gestionar proyectos de administrador

Análisis

El administrador puede visualizar todos los proyectos registrados en el sistema, así como registrar, modificar o eliminar un proyecto.

El administrador, tiene la facultad y poder de:

Registrar Proyecto: Para registrar los proyectos se le solicita al administrador ingresar la siguiente información (atributos).

- **Clave:** Con el fin de tener unicidad de registros en la base de datos.
- **Nombre:** Con el fin de identificar los módulos.
- **Fecha de inicio:** Que identifica la fecha de inicio del proyecto.
- **Fecha de término:** Que identifica la fecha fin del proyecto prevista.
- **Fecha de inicio programada:** Que identifica la fecha de inicio programada del proyecto.
- **Fecha de término programada:** Que identifica la fecha fin del proyecto programada.
- **Líder del Proyecto:** Para la creación de un proyecto debe de haber al menos un colaborador registrado.
- **Descripción** Para especificar las características y contenido del proyecto.
- **Contraparte:** Para identificar al cliente o solicitante del proyecto.
- **Presupuesto:** Con el fin de identificar el presupuesto destinado al proyecto.
- **Estado del Proyecto:** Con el fin de identificar en que estado se encuentra el proyecto.

Se puede registrar un proyecto siempre y cuando exista al menos un colaborador registrado, así como la información referente a los estados del proyecto.

Una vez registrado el proyecto se podrán gestionar los Términos del glosario, Entidades, Reglas de negocio, Mensajes y Actores.

Modificar Proyecto: Dentro del sistema existe la posibilidad de modificar la información de un proyecto previamente registrado, siempre y cuando el proyecto se encuentre en estado .^{En} negociación.^o

Inicio"

Eliminar Proyecto: El sistema debe permitir al administrador eliminar el registro de un proyecto.

7.4. Sprint 3: Gestión de Proyectos de Colaborador

7.4.1. Gestionar proyectos de colaborador

Análisis

El actor puede visualizar los proyectos en los que participa, este es el punto de acceso para gestionar: módulos, términos del glosario, entidades, reglas de negocio, mensajes y actores, así como para descargar el documento de análisis y en caso de ser líder, elegir a los colaboradores.

Elegir Colaboradores: El líder de análisis tiene la facultad de seleccionar a aquellas personas que colaborarán en el proyecto. Para que una persona pueda ser colaboradora tiene que haberse registrado previamente en el módulo de registro de personal.

7.5. Sprint 4: Gestión de Módulos

7.5.1. Gestionar módulos

Análisis

Para facilitar la construcción de sistemas de software, la estrategia de Divide y Vencerás es una técnica imprescindible, la cual se basa en la descomposición de un problema en subproblemas de su mismo tipo, lo que permite disminuir la complejidad de los mismos. La manera en la que se descompone un sistema es a través de sus módulos, es por eso que se le proporciona al actor un mecanismo para llevar el control de los módulos de un proyecto.

Registrar Módulo: Para registrar los módulos del proyecto se le solicita al actor ingresar la siguiente información (atributos).

- **Clave:** Con el fin de tener unicidad de registros en la base de datos.
- **Nombre:** Con el fin de identificar los módulos.
- **Descripción** Para especificar las características y contenido del módulo.

Modificar Módulo: Dentro del sistema existe la posibilidad de modificar los datos de un módulo previamente registrado, a excepción de su clave.

Eliminar Módulo: El sistema debe permitir al actor eliminar el registro de un módulo siempre y cuando no existan referencias al contenido del módulo a eliminar desde elementos de algún otro módulo.

7.6. Sprint 5: Gestión de Términos de glosario

7.6.1. Gestionar términos del glosario

Análisis

En un proyecto es de suma importancia que se tenga una lista de palabras y expresiones clasificadas de un texto con su respectivo significado. La correcta definición conceptual de los términos de negocio en el sistema se hace imprescindible para comprenderlo.

El colaborador, tiene la facultad y poder de:

Registrar Término: Para registrar la información de un término se le solicita al actor ingresar la siguiente información (atributos).

- **Nombre:** Con el fin de identificar los términos.
- **Descripción:** Para especificar las características y contenido del término.

Modificar Término: Dentro del sistema existe la posibilidad de modificar los datos de un término previamente registrado.

Eliminar Término: El sistema debe permitir al actor eliminar el registro de un término existente.

Consultar Término: El sistema debe permitir al actor consultar la información de un término existente.

7.7. Sprint 6: Gestión de Entidades

7.7.1. Gestionar Entidades

Análisis

Cuando se habla sobre la construcción de un proyecto de software, se habla también de la interacción de diversas entidades para lograr un fin. Una entidad es un objeto exclusivo único en el mundo real que se está controlando. Una entidad puede referirse a una persona, organización, tipo de objeto o concepto sobre los que se almacena información. Al ser un elemento clave participe dentro del desarrollo de software es importante contar con un catálogo de estos elementos.

El colaborador, tiene la facultad y poder de:

Registrar Entidad: Para registrar la información de una entidad se le solicita al colaborador ingresar la siguiente información (atributos).

- **Clave:** Que permitirá distinguir que el Elemento es una Entidad en una palabra corta.
- **Número: (Generado Automáticamente)** Para llevar un control interno de la cantidad de elementos en el proyecto.
- **Nombre:** Nombre que identificará a la Entidad en una frase o enunciado.

Modificar Entidad: Dentro del sistema existe la posibilidad de modificar los datos de una entidad previamente registrada.

Eliminar Entidad: El sistema debe permitir al colaborador eliminar el registro de una entidad existente.

Consultar Entidad: El sistema debe permitir al colaborador consultar la información de una entidad existente.

7.8. Sprint 7: Gestión de Atributos

7.8.1. Gestionar Atributos

Análisis

Al considerar la inclusión de entidades como elementos participantes dentro del Proyecto, no se deben perder de vista los atributos propios de cada Entidad. Un Atributo es una característica o rasgo de una Entidad que la describe, por ejemplo, el tipo de entidad Persona tiene el atributo Fecha de nacimiento.

El colaborador, tiene la facultad y poder de:

Registrar Atributo: Para registrar la información de un atributo se le solicita al colaborador ingresar la siguiente información (atributos).

- **Nombre:** Que identificará al Atributo en la Entidad en una frase o enunciado.
- **Descripción:** Texto que describirá al Atributo en uno o más párrafos.
- **Obligatorio:** Bandeara que indicará si el Atributo es obligatorio o no lo es.
- **Longitud:** Número que describirá la longitud máxima del Atributo.

Modificar Atributo: Dentro del sistema existe la posibilidad de modificar los datos de un atributo previamente registrado.

Eliminar Atributo: El sistema debe permitir al colaborador eliminar el registro de un atributo existente.

Pruebas Realizadas

8.1. Pruebas Dinámicas

Static and dynamic testing are both needed to cover the range of products we need to test.

8.1.1. Diseño de Pruebas

MATRICES DE PRUEBA

Las matrices de pruebas son una herramienta que detalla el resultado de la ejecución de pruebas en comparación con cada una de las características y funcionalidades de los módulos del sistema. Están basadas en el documento de análisis, específicamente en el documento de casos de uso.

La ejecución de pruebas se lleva a cabo una vez terminado el desarrollo de cada uno de los sprints estructurados en la metodología.

El objetivo de las matrices de prueba es validar y verificar a través de ciclos de pruebas que el producto cumpla con lo especificado en las etapas precedentes, en caso de que no se cumpla con las funcionalidades estipuladas o existan mejoras en su implementación se reporta una serie de defectos y sugerencias para que desarrollo se encargue de corregirlos; una vez que se solucionaron los problemas se ejecuta un ciclo de confirmación para comprobar que efectivamente se han corregido los defectos y que no se ha alterado ninguna otra parte del sistema.

A continuación se ilustra la estructura de las matrices de prueba que se diseñaron para registrar el resultado de su ejecución en los ciclos correspondientes.

Prueba:	PF1-Gestionar Módulos		Número de pasos	
Tipo:	Sistema		Pasos a evaluar	3
Nivel:	Fábrica y aceptación		Pasos evaluados	
Propiedades:	Funcional y robustez		Pasos por evaluar	
Número de Casos a probar:	4			
Elaboró el guión de pruebas:	Esteban Martínez		Mensajes:	Pantallas:
Ejecutó el guión de pruebas:	Giselle Olvera		MSG2 No existe información	IU5 Gestionar proyectos de Colaborador
Caso de prueba:	CU5 Gestionar Módulos			
Actores:	Líder de análisis, Analista			

Figura 8.1: Ejemplo del encabezado de la matriz de pruebas

						PRIMER CICLO DE PRUEBA: 14/09/2019		
Número de paso de la prueba	Descripción del paso	Estados	Datos	Catálogos	Salida esperada	Salida obtenida Ok o descripción de la salida en caso de ser diferente a la esperada	Resultado Ok o Fail	Observaciones Descripción detallada de lo que se observa y de la prueba realizada
1	Inicia sesión como Líder de análisis o analista		(USR: este_p@hotmail.com, PSW: pruebatt)		Se muestra la pantalla IU5 con la opción de "Entrar al Proyecto"	ok	ok	
2	Trayectoria Principal				Muestra la tabla "Módulos" con todos los registros de los módulos asociados al proyecto seleccionado en la pantalla IU4	ok	fail	Mensaje incorrecto
3	Trayectoria Alternativa GM-A				Muestra el mensaje MSG2 en la pantalla IU2	fail	fail	

Figura 8.2: Ejemplo de la estructura de la matriz de pruebas

REPORTE DE PRUEBAS SPRINT 1 - CICLO 1

Las pruebas contempladas para el primer ciclo de pruebas abarcan los Casos de Uso del Sprint 1, Los cuales comprenden las siguientes gestiones:

- CU1 Iniciar sesión.
- CU2 Gestionar proyectos de Administrador.

- CU3 Gestionar Colaboradores.
- CU4 Gestionar Proyectos de Colaborador.

Se realizaron pruebas dinámicas de sistema, con técnicas de caja negra.

Base de prueba:

- Casos de Uso
- Especificaciones de requisitos del sistema y software.
- Sistema y manual de usuario.

Objeto de prueba:

- Sistema de software

Los resultados finales del primer ciclo de prueba arrojaron los siguientes datos:

SEVERIDAD					
ESTATUS	Crítica	Alta	Media	Baja	Total
Encontrado	1	6	3	3	13
TOTAL DEFECTOS CICLO 1					13

Severidad	Total
Crítico	1
Alta	6
Media	3
Baja	3
Total	13

Tipo Defecto	Crítica	Alta	Media	Baja	Total
Bloqueante	0	0	0	0	0
Datos	0	0	0	0	0
Diseño	0	0	0	0	0
Funcional	1	6	2	0	9
Requerimientos	0	0	0	1	1
Redacción	0	0	0	1	1
Mejora/Observación	0	0	1	1	2
TOTAL TIPO DE DEFECTO POR CICLO					13

Figura 8.3: Informe de defectos Sprint 1 Ciclo 1

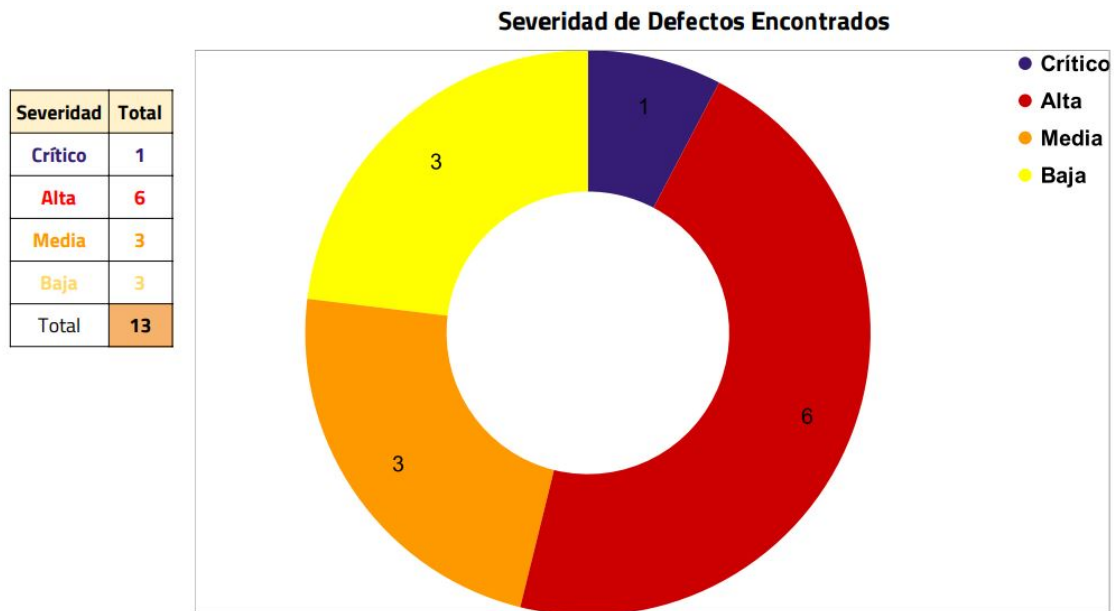


Figura 8.4: Gráfica de defectos por severidad Sprint 1 Ciclo 1

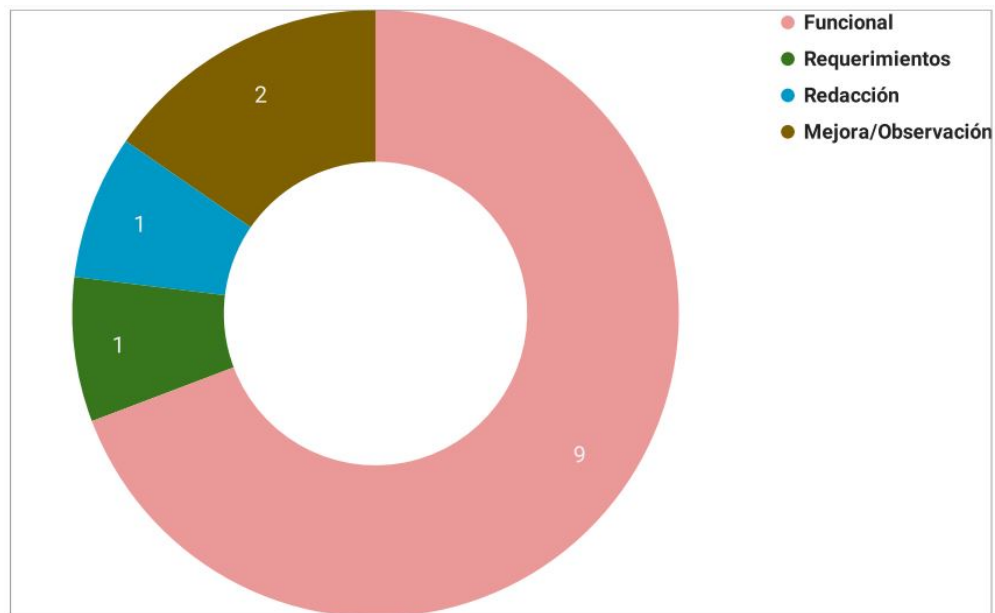


Figura 8.5: Gráfica de defectos por tipo de defecto Sprint 1 Ciclo 1

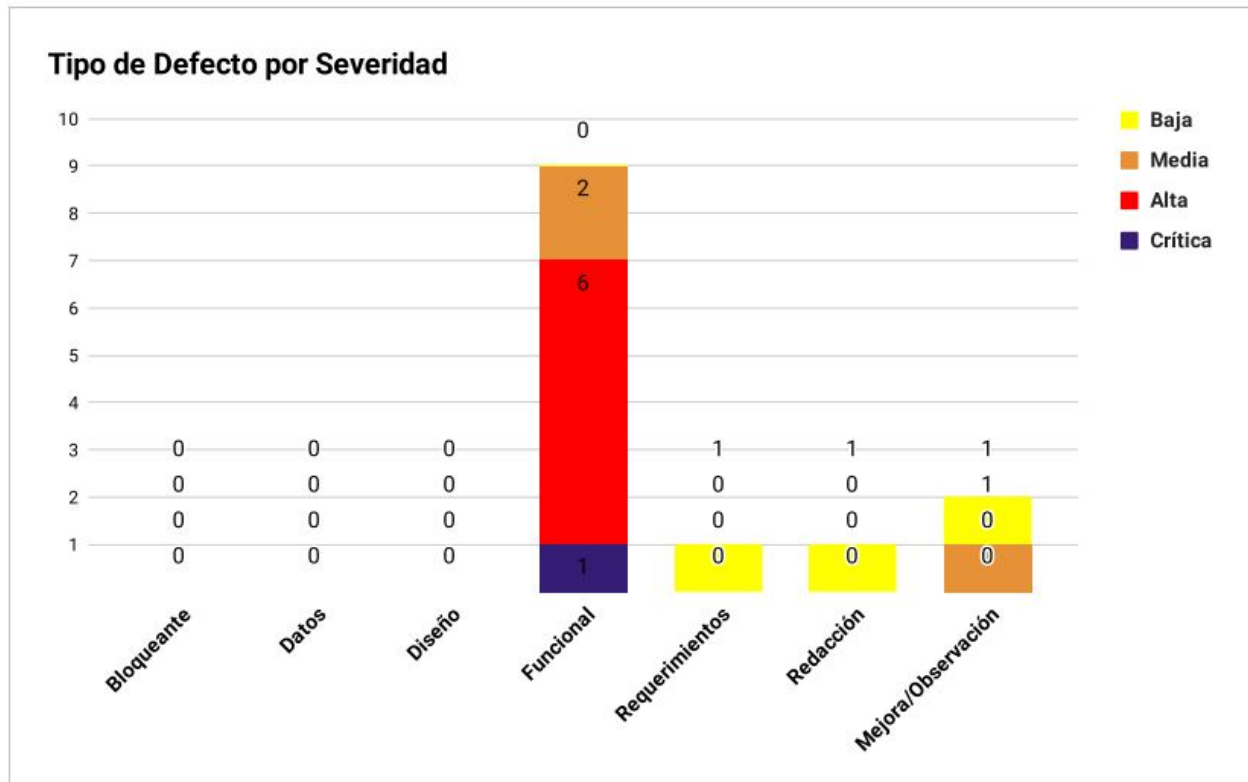


Figura 8.6: Gráfica de tipo de defectos por severidad Sprint 1 Ciclo 1

Se corrigieron los defectos encontrados en el primer ciclo.

Bibliografía

- [1] Rui, K. Butler, G. (2003, April 21). Refactoring use case models: the metamodel [Online]. Available: <https://dl.acm.org/citation.cfm?id=783140>
- [2] Shuang, L. Sun, L. (2014, September 19). Automatic early defects detection in use case documents [Online]. Available: <https://dl.acm.org/citation.cfm?id=2642969>
- [3] J. Lee. (1999, August). Analyzing user requirements use cases a goal driven approach. [Online]. Available: <https://ieeexplore.ieee.org/document/776956>
- [4] L. Julijana. (2007, August). “Information Systems Modeling with Use Cases” IEEE Computer [Online]. Available: <https://ieeexplore.ieee.org/document/4283759>
- [5] Jason Gorman, J. G. (2007, 9 marzo). 10 Common Use Case Pitfalls. Recuperado 23 abril, 2018, [Online]. Available: <http://codemanship.co.uk/parlezuml/blog/?postid=364>
- [6] Susan Lilly [2002, August]. “Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases” [Online]. Available: <https://ieeexplore.ieee.org/document/787547>
- [7] Universidad Michoacana de San Nicolás de Hidalgo. (2014): “Competitividad y factores de éxito en empresas desarrolladoras de software” [Online]. Available: <https://dialnet.unirioja.es/servlet/articulo?codigo=510>
- [8] Mr. Vikas S. Chomal, Dr. Jatinderkumar R. Saini, (2014). “Significance of Software Documentation in Software Development Process“ / (7 páginas). International Journal of Engineering Innovation and Research.
- [9] Pressman, Roger. (2010). Ingeniería de Software. Un enfoque práctico / 7 ED.(777 páginas). USA: McGraw-Hill Interamericana.

-
- [10] B. Imran S. and H. Irfan, "ÜCD-generator - a LESSA application for use case design - IEEE Conference Publication", Ieeexplore.ieee.org, 2007. [Online]. Available: <https://ieeexplore.ieee.org/document/4381333>.
- [11] G. Booch, J. Rumbaugh, I. Jacobson, J. García Molina and J. Saez Martínez, *El lenguaje unificado de modelado*, 2nd ed. Madrid: Pearson Educación, 2010.
- [12] D. West, "Use Cases Considered Valuable (but Optional) For Lean/Agile Requirements Capture", InfoQ, 2010. [Online]. Available: <https://www.infoq.com/news/2009/02/Use-Cases-Valuable-But-Optional>.
- [13] D. González, "Industria Mexicana del Software. Un estudio en cifras.", SG Buzz, 2005. [Online]. Available: <https://sg.com.mx/revista/9/industria-mexicana-cifras>. [Accessed: 20- Apr- 2019].
- [14] J. Gómez, "Métodos de Medición en Puntos Función (I): IFPUG FPA", El Laboratorio de las TI, 2014. [Online]. Available: <https://www.laboratorioti.com/2013/01/16/metodos-de-medicion-en-puntos-funcion-i/>.
- [15] Ganesh Krishnamurthy, "CASE Tools Adoption and Relevance", University of Missouri–St. Louis [Online]. Available: <http://www.umsl.edu/~sauterv/analysis/F08papers/View.html>
- [16] Annette L. du Plessis, "A method for CASE tool evaluation", *Information and Management*, Volume 25, Issue 2, August 1993, Pages 93-102
- [17] Erich Gamma, "Patrones de diseño: elementos de software orientado a objetos reusable", Pearson Educación, 2002. Addison-Wesley professional computing series.
- [18] IBM Knowledge Center, "Patrón de diseño de modelo-vista-controlador"[Online]. Available: <https://www.ibm.com/support/knowledgecenter/es>
- [19] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language*. Oxford University Press, New York, 1977.
- [20] Glenn E. Krasner and Stephen T. Pope. "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80." *Journal of Object-Oriented Programming*, 1(3):26–49, August/-September 1988.
- [21] BBVA-TRANSFORMACIÓN DIGITAL, "Metodología 'scrum': ¿Qué es un 'sprint'?", [Online]. Available: <https://www.bbva.com/es/metodologia-scrum-que-es-un-sprint/>
- [22] Proyectos Agiles, "Lista de tareas de la iteración (Sprint Backlog)", [Online]. Available: <https://proyectosagiles.org/lista-tareas-iteracion-sprint-backlog/>
- [23] Clemente Ruiz Durán, Michael Piore Andrew Schrkarn, "Los retos para el desarrollo de la industria del software", [Online]. Available: <http://revistas.bancomext.gob.mx/rce/magazines/87/1/Ruiz-Schrank.pdf>

-
- [24] Universidad Católica de los Angeles Chimbote-PERÚ, "Metodología de Desarrollo de Software"[Online].
Available: <https://www.uladech.edu.pe/images/stories/universidad/documentos/2018/metodologia-desarrollo-software-v001.pdf>
- [34] ISTQB (International Software Testing Qualifications Board), "Foundations of Software Testing"
- [25] (<https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/data-access.html#transaction>)
- [26] (<https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/data-access.html#transaction-declarative>)
- [27] <https://struts.apache.org/birdseye.html>
- [28] <https://struts.apache.org/primer.html#mvc>
- [29] <https://struts.apache.org/core-developers/interceptors.html#order-of-interceptor-execution>
- [30] https://www.researchgate.net/profile/Praveen_Gupta25/publication/49619227_MVC_Design_Pattern_for_the_
- [31] (https://link.springer.com/chapter/10.1007/978-1-4302-2370-2_20)
- [32] (<https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/n-tier>)
- [33] (<https://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch04s04.html#beans-factory-scopes-singleton>)