UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN

INGENIERIA DE SOFTWARE APLICADA EN LA ELABORACION DE HERRAMIENTAS E-LEARNING.

TESIS

QUE PARA OBTENER EL TÍTULO DE

Licenciado en informática

P R E S E N T A

Antonio Hernández Oropeza

DIRECTORA DE TESIS

MGTI. Rosalba Nancy Rosas Fonseca

Cuautitlán Izcalli, Estado de México, 2022

VNIVERSIDAD NACIONAL AVTNOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO TITULACIÓN

U. N. A. M.
FACULTAD DE ESTUDIOS

ASUNTO: VOTO APROBATORIO

DR. DAVID QUINTANAR GUERRERO
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: DRA. MARÍA DEL CARMEN VALDERRAMA BRAVO
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: **Trabajo de Tesis**

INGENIERIA DE SOFTWARE APLICADA EN LA ELABORACION DE HERRAMIENTAS E-LEARNING

Que presenta el pasante: **Antonio Hernández Oropeza**
Con número de cuenta: **314332002** para obtener el Título de: **Licenciado en Informática**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 8 de abril de 2022.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Mtro. Rogelio Moisés Sánchez Arrastio	
VOCAL	Dr. Leonel Gualberto López Salazar	
SECRETARIO	M.G.T.I. Rosalba Nancy Rosas Fonseca	
1er. SUPLENTE	Mtra. Maria Guadalupe Vázquez Salazar	
2do. SUPLENTE	Mtra. Judith Mayte Flores Pérez	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

MCVB/ntm*

Dedicatorias

A mí:

Con la conclusión de esta etapa en tu vida, espero que, como hoy, siempre tengas presentes la lista de sueños y objetivos que tienes desde que eras niño, que siempre recuerdes que un “No puedo” no debe estar en tu vocabulario y que el fracaso es una parte del éxito, deseo que jamás pierdas el entusiasmo por aprender cosas nuevas, deseo que jamás tengas miedo al cambio y deseo que, como hoy, tengas siempre las ganas de crecer como persona y profesionalmente.

A mi familia:

Cada día desde que era niño ustedes me han acompañado en cada paso en mi vida, y tengo la fortuna de que hoy después de tanto tiempo podamos compartir este logro juntos, espero seguir teniendo la fortuna de compartir más momentos con ustedes a lo largo de mi vida.

A Fernanda:

Te has convertido en mi acompañante de vida; durante mucho tiempo, hemos pasado tantas cosas juntos, espero que este y otros logros los pueda compartir contigo.

A la Mtra Nancy:

Mas que una profesora de asignatura, se convirtió en mi guía para poder culminar esta etapa en mi vida, agradezco tanto la dedicación, paciencia y amistad que me ha brindado.

Contenido

Resumen.

A través de esta tesis, se explican de manera amplia todas las etapas que involucran al desarrollo de software, desde la planeación, el diseño, desarrollo, pruebas y despliegue además de evaluar y entender todas las herramientas que se encuentran alrededor de software tanto para la documentación, como para su elaboración, un ejemplo de ello es el uso de herramientas CASE, herramientas de prototipado, generador de documentación o generador de código, es importante entender la importancia de estas herramientas desde incluso la planeación del desarrollo, luego se documenta la parte del inicio del desarrollo y aspectos que se deben de entender y al mismo tiempo se deben conocer como las metodologías en general, y con un enfoque actual a las metodologías agiles, ya que es necesario entenderlas y no subestimar su importancia, porque al final una metodología ágil como SCRUM hará que el proyecto tenga la capacidad de evolucionar, hará que se pueda tener una gestión transparente y ágil al momento de tener nuevos requerimientos.

Otro punto que se busca destacar con esta investigación es el uso y el empleo de buenas practicas de desarrollo, así como la implementación patrones de diseño y arquitecturas, tanto como pruebas manuales y automatizadas, si bien es cierto actualmente en la industria el perfil del ingeniero de software ha cambiado y se a centrado en una pila de conocimientos los cuales no solo involucran la escritura de código, sino también la prueba de este, el despliegue y una serie de buenas prácticas, incluso para documentar su estructura, tanto es así, que incluso existen roles específicos de “Code Documentor”, precisamente porque la industria demanda esas habilidades, incluso el rol de Dev Ops surge precisamente por la necesidad de tener una persona que este involucrada en el área de desarrollo de la misma manera en como esta involucrado en el área de operaciones, entonces, es necesario que los alumnos conozcan esta parte de la industria para que su formación profesional cubra estas necesidades.

Capítulo I:

Introducción

El desarrollo de software ha incrementado en los últimos años , sin embargo esto último no significa que sea maquila, como sabemos, el software es una obra intelectual, y si bien se pueden hacer prototipos, o plantillas, no existe una forma de hacer un molde general para software, tan solo la construcción del mismo ya implica muchas problemáticas, y como en otros procesos, previo a la construcción se debe llevar a cabo una planeación estratégica, que de manera muy sintetizada en desarrollo de software es levantar requerimientos, hacer diagramas, utilizar una metodología e implementar , aunque en texto se puede escribir fácil, la realidad es que en el gremio muchos profesionistas incluso llegan a precipitarse y empiezan por “tirar código”, desde luego es una problemática que también se da mucho en principiantes y novatos, una analogía sencilla para entender la problemática es pensar que podemos llegar a construir un edificio sin hacer levantamientos, estudio del terreno, planos etc; eso mismo pasa en el desarrollo de software, por ello se tiene como propósito que los estudiantes (principalmente) utilicen esta tesis como una guía de iniciación, con un caso práctico y sobre todo la experiencia de lo que se debe hacer y lo que no se debe hacer.

Si, ya existen muchas tesis o libros referentes al tema, afortunadamente internet nos permite reducir esa brecha del conocimiento, sin embargo, considero preciso elaborar una tesis que tenga como finalidad demostrar la cantidad de errores funcionales que se pueden cometer a la hora de desarrollar software, y que incluso en el camino del desarrollo yo tuve, de hecho, es necesario apuntar que esta tesis precisamente surge de la detección de la carencia de habilidades y/o conocimientos incluso ya estando en séptimo semestre de la licenciatura en informática, por lo tanto es una recopilación de que hicimos como equipo, que debimos hacer, en que tuvimos éxito y en que fracasamos, esperando que los estudiantes puedan basarse en esta lista para no cometer los mismos errores a la hora de desarrollar sus proyectos.

1.1 Objetivos:

1.1.1 General:

- Como objetivo general se tiene Generar una propuesta para que los estudiantes interesados en desarrollar software tomen en cuenta las metodologías, patrones de diseño, tecnologías, guías y estructuras existentes con la finalidad de generar una cultura de desarrollo con buenas practicas y mas ordenada, poniendo como caso de estudio el desarrollo de una plataforma de E-Leaning.

1.1.2 Específicos:

- Documentar la estructura de un proyecto desarrollado en la UNAM, con el fin de que la comunidad pueda comprender las etapas y procesos de la ingeniería de software a través de un caso práctico.
- Crear un trabajo que sirva como un extracto y/o recopilación de información útil a través de la cual el alumno tenga la capacidad de aplicarla en el desarrollo de sus proyectos durante la carrera y en el ámbito laboral.
- Concientizar a los alumnos sobre la importancia de apoyarse de la ingeniería de software y de procesos específicos para el desarrollo de proyectos de software.

1.1 Antecedentes

Generales:

Tal como lo menciona el CONAIC (Consejo Nacional de Acreditación en Informática y computación A.C), la ingeniería de software es una disciplina profesional que al año 2020 tiene apenas 44 años de existencia con programas de posgrado en Estados Unidos y en 1987 con programas de posgrado en reino unido, sin embargo fue hasta el 2004 cuando en México se ofreció por primera vez un programa educativo a nivel licenciatura en la Universidad Autónoma de Yucatán, ofreciendo el título de Ingeniero de Software.

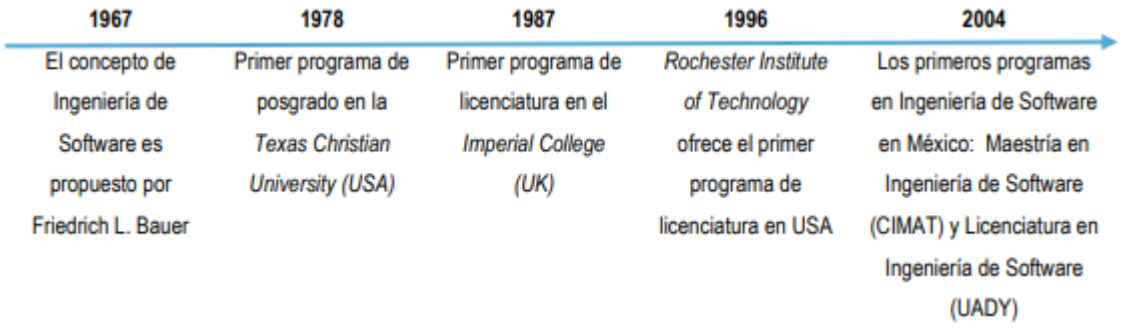


Figura 1. Primeros Programas Educativos en IS.

Específicos:

Actualmente en la Licenciatura en Informática impartida en la Facultad de Estudios Superiores Cuautitlán se imparten al menos 4 materias relacionadas con la ingeniería de software, específicamente a la gestión de proyectos, las cuales son: Informática III. Análisis y Diseño de Sistemas I , Informática IV. Análisis y Diseño de Sistemas II, Informática V. Industria del Software y Laboratorio de Sistemas de Información en 3er,4to,5to y 7mo semestre respectivamente.

1.2 Justificación

Una de las problemáticas que enfrenta el egresado de la licenciatura de informática es el precario conocimiento de la ingeniería de software, ya que una práctica muy común en los alumnos es desarrollar software sin seguir alguna metodología, el problema es que al ser una carrera de TI el mundo laboral es muy competente, y no tener dichas prácticas de desarrollo es perjudicial para el desarrollo de software, es aquí en donde surge la necesidad de crear un recurso el cual pueda ser consultado por los alumnos para tener un marco de referencia practico sobre la ingeniería de software, mientras se documenta la elaboración de un módulo de Ciudad Virtual.

1.3 Metodología

Para dicha tesis emplearemos una metodología de tipo hipotético-deductivo ya que primero observaremos el fenómeno a estudiar; que en este caso es el uso deficiente o nulo de la ingeniería de software posteriormente formularemos una hipótesis la cual se basará en el conocimiento previo o empírico, seguido de esto se genera una serie de deducciones o consecuencias que van de la mano con la hipótesis para finalmente ser comprobada o rechazada con la experiencia.

1.4 Hipótesis

Demostrar a través del desarrollo de una plataforma de e-learning las deficiencias del alumno de la licenciatura en informática para trabajar en proyectos de desarrollo de software, y a partir de esto crear una guía sintetizada para que se les facilite a los alumnos utilizar metodologías y procesos de ingeniería de software y con la cual ellos puedan:

- Hacer un correcto levantamiento de requerimientos
- Hacer un análisis y diseño de manera estructurada
- Hacer una implementación ordenada
- Realizar las pruebas y documentación correspondiente

1.5 Marco teórico

Bauer (1972) presento a la ingeniería del software como el establecimiento y uso de principios técnicos sanos para obtener económicamente el software que es confiable y trabaja en verdaderas máquinas eficazmente.

Bohem (1976) presentó a la Ingeniería del Software como la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollarlos, operarlos y mantenerlos

Zelkovitz (1978) describió a la Ingeniería del Software como el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas de software.

En general podemos definir a la ingeniería de software como una disciplina con técnicas, reglas y metodologías que permiten la planificación, el diseño y la construcción de software como aplicaciones móviles, de escritorio, páginas web; entender el concepto de ingeniería de software es de suma importancia a la hora de emplear técnicas y metodologías, porque eventualmente muchas de estas cosas no se pueden emplear correctamente porque el equipo de desarrollo suele desconocer muchas etapas o herramientas de desarrollo.

Capítulo II:

Ingeniería de software

2.1 Herramientas CASE.

Las herramientas CASE (Computer Aided Software Engineering o Ingeniería de Software Asistida por Computadora en español) son diversas aplicaciones o programas informáticas que tienen como finalidad aumentar la productividad y eficiencia en el desarrollo de software de esta manera reducir el tiempo y dinero de producción, estas herramientas pueden servir en cualquier etapa del desarrollo de software, en tareas como realizar el diseño de un proyecto, cálculo de costo, implementación de código, compilación automática, documentación y detección de errores entre otras cosas.

De otra manera también se pueden definir como el conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.

Clasificación.

Como tal no existe una clasificación universal para estas herramientas, de hecho, hay varias clasificaciones y entre ellas destacan:

Las plataformas que soportan.

Las frases del ciclo de vida del desarrollo de sistemas que cubren.

La arquitectura de las aplicaciones que producen.

Su funcionalidad.

Herramientas integradas, I-CASE (Integrated CASE, CASE integrado): abarcan todas las fases del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE workbench.

2. Herramientas de alto nivel, U-CASE (Upper CASE - CASE superior) o front-end, orientadas a la automatización y soporte de las actividades desarrolladas durante las primeras fases del desarrollo: análisis y diseño.

3. Herramientas de bajo nivel, L-CASE (Lower CASE - CASE inferior) o back-end, dirigidas a las últimas fases del desarrollo: construcción e implantación.

4. Juegos de herramientas o Tools-Case, son el tipo más simple de herramientas CASE. Automatizan una fase dentro del ciclo de vida. Dentro de este grupo se encontrarían las herramientas de reingeniería, orientadas a la fase de mantenimiento

Entre los principales diagramas y modelos que destacan en cuanto a uso de herramientas CASE:

- Diagrama de flujo de datos.
- Modelo entidad-relación.
- Historia de la vida de las entidades.
- Diagrama de estructura de datos.
- Diagrama de estructura de cuadros.
- Técnicas matriciales.

Algunas de las características de los diagramas y modelos son:

- Número máximo de niveles para poder soportar diseños complejos.
- Dibujos en formato libre con la finalidad de añadir comentarios, dibujos, información adicional para aclarar algún punto concreto del diseño.
- Control sobre el tamaño, fuente y emplazamiento de los textos en el diagrama.
- Posibilidad de deshacer el último cambio, facilitando que un error no conlleve a perder el trabajo realizado.
- Inclusión de pseudocódigo, que servirá de base a los programadores para completar el desarrollo de la aplicación.

Herramientas de prototipado.

El objetivo principal de esta herramienta es poder mostrar al usuario, desde los momentos iniciales del diseño, el aspecto que tendrá la aplicación una vez desarrollada. Ello facilitará la aplicación de los cambios que se consideren necesarios, todavía en la fase de diseño.

Generador de código.

Es uno de los componentes indispensables de las herramientas CASE, algunas de las características más importantes de este componente son: el o los lenguajes de programación para los que se genera código y el alcance de la generación del cuerpo del programa.

Generador de documentación.

El módulo generador de la documentación se alimenta del repositorio para transcribir las especificaciones allí contenidas, entre las principales características se encuentran:

- Generación automática a partir de los datos del repositorio, sin necesidad de un esfuerzo adicional.
- Combinación de información textual y gráfica, lo que hace más fácil su comprensión.
- Ayuda de tratamiento de textos. Facilidad para la introducción de textos complementarios a la documentación que se genera de forma automática.
- Interface con otras herramientas: procesadores de textos, editores gráficos, etc.

2.2 Actividades fundamentales del desarrollo de software.

Actualmente en el desarrollo de software existen muchas actividades las cuales forman parte fundamental de la ingeniería de software, como lo hemos visto anteriormente parte del desarrollo adecuado de estos métodos y técnicas son elementales a la hora de que un proyecto tenga éxito o no, Si bien es cierto existen de acuerdo al tipo de proyecto diferente actividades que son fundamentales para el desarrollo de los proyectos, en este caso con la finalidad de explicar en gran medida muchas actividades que involucra, tomaremos las actividades que generalmente se emplean en la mayor cantidad de proyectos, y eventualmente a la hora de abordar mi proyecto explicare cuales, si utilice, y cuales debí utilizar; entrando en materia, generalmente las actividades esenciales consisten en:

Figura 1.1 Participantes en el desarrollo de software (Pleeger 2002)

2.2.1 Recopilación y análisis de requerimientos.

La recopilación de requerimientos es probablemente una de las actividades que lejos de ser fundamental es crucial a la hora de inicia la planeación de un proyecto de software, ya que es la transformación de las necesidades de los clientes, ya sean hablados o escritos, se convierten en especificaciones precisas, no ambiguas y completas; al mismo tiempo se encarga de establecer y mantener los cambios de requerimientos entre clientes y equipo de desarrollo, En otras palabras, los requerimientos identifican el qué del sistema, mientras que el diseño establece el cómo del sistema.

Un levantamiento de requerimientos correcto nos permite:

- Mantener una estructura en las necesidades del proyecto.
- Disminuir costos y retrasos.
- Mejorar la calidad del software.
- Mejorar la comunicación.
- Evitar rechazos de usuarios finales.

2.2.2 Especificaciones del software

Posteriormente en el análisis de requerimientos, que nos ayuda a identificar las necesidades del cliente, crearemos las especificaciones del software que son un conjunto de casos que describen la interacción que debe tener el usuario con el software, hasta este paso se puede seguir utilizando un lenguaje “informal”, ya que es un proceso del cliente y el equipo de desarrollo, por lo cual debe ser un lenguaje digerible para las dos partes, y en el cual debe existir una comunicación funcional y efectiva.

Según el estándar de la IEEE 830-1998 las características recomendadas que debe tener las especificaciones del software son:

- Completo
    - Es decir, los requisitos deben estar directamente relacionados a la funcionalidad, desarrollo, restricciones del diseño y las interfaces externas.
- Consistente
    - Debe existir consistencia entre los requerimientos y entre otros documentos sobre especificaciones de sistemas.
- Inequívoco
    - Se considera inequívoco si y solo si cada requisito declarado tiene una interpretación única, en caso de que el termino lo amerite debe ser creado un glosario, debido a las trapas que lleva el lenguaje natural (humano), como la ambigüedad, entonces se recomienda hacer una representación ,estas herramientas comúnmente apoyan 3 categorías : el objeto, proceso, conductual; como sabemos los objetos hacen referencia a las cosas en el mundo real con sus respectivos atributos, el proceso indica una jerarquía de funciones con un determinado fin y por otro lado el conductual describe la conducta externa del sistema .
- Correcto
    - Una especificación de requerimiento de software es correcta solo si se la especificación declarada se encuentra desarrollada en el sistema, aunque no existe una herramienta que asegure exactitud, sin embargo, el cliente o usuario paralelamente puede verificar si la especificación realmente refleja las necesidades.
- Trazable
    - Se debe verificar que tenga una forma de rastreo dicho requerimiento, para los fines que sean necesarios en el futuro.

- Prioridad
    - Debe existir una organización jerárquica la cual organice los requisitos en relevancia, según el giro y/o necesidades del negocio, comúnmente clasificándolos en esenciales, condicionales y opcionales.
- Modificable
    - Es modificable solo si la estructura y el estilo son tales que se les puede hacer cualquier cambio a los requisitos fácilmente, completamente y de forma consistente mientras conserva la estructura y estilo, con ello no es posible que pierda la coherencia, y que no sea redundante.
- Identificable
    - Se puede identificar si el origen es claro y facilita las referencias de cada requisito en el desarrollo futuro o documentación del mismo.

Una manera de agrupar estos requerimientos es a través de fichas las cuales deben incluir aspectos como:

- Numero de requisito
- Nombre de requisito
- Descripción
- Tipo
- Fuente del requisito
- Prioridad del requisito

Número de requisito	!!br0ken!!RF01		
Nombre de requisito	Registrar ingreso de paciente de urgencia		
Descripción	El usuario a cargo del sistema podrá registrar el ingreso de pacientes, madres embarazadas, a la unidad de maternidad.		
Tipo	☒ Requisito	☐ Restricción	
Fuente del requisito	Documento, "Planteamiento del Problema".		
Prioridad del requisito	☐ Alta/Esencial	☒ Media/Deseado	☐ Baja/ Opcional

Figura 1.2 Ejemplo de formato para recopilar requerimientos (Inasoft 2015)

Parte de esto requiere de la plena identificación de los tipos de requerimiento, entre los cuales existen:

1. Requisitos de usuarios:

   Son los requisitos “directos”, es decir las necesidades directas que expresa el cliente.

2. Requisitos del sistema

   Son aquellos requerimientos que especifican los componentes del sistema necesarios para cumplir las tareas.

3. Requisitos funcionales

   Son funcionalidades que debe hacer el sistema al final del desarrollo

4. Requisitos no funcionales

   Define propiedades o restricciones de las funcionalidades.

Entre las principales restricciones podemos encontrar:

a) Las políticas reguladoras.

b) Las limitaciones del Hardware.

c) Las Interfaces a otras aplicaciones.

d) El funcionamiento Paralelo.

e) Las funciones de la Auditoría.

f) Las funciones de Control.

g) Los requisitos de lenguaje.

h) Los protocolos Señalados (por ejemplo, XON-XOFF, ACK-NACK).

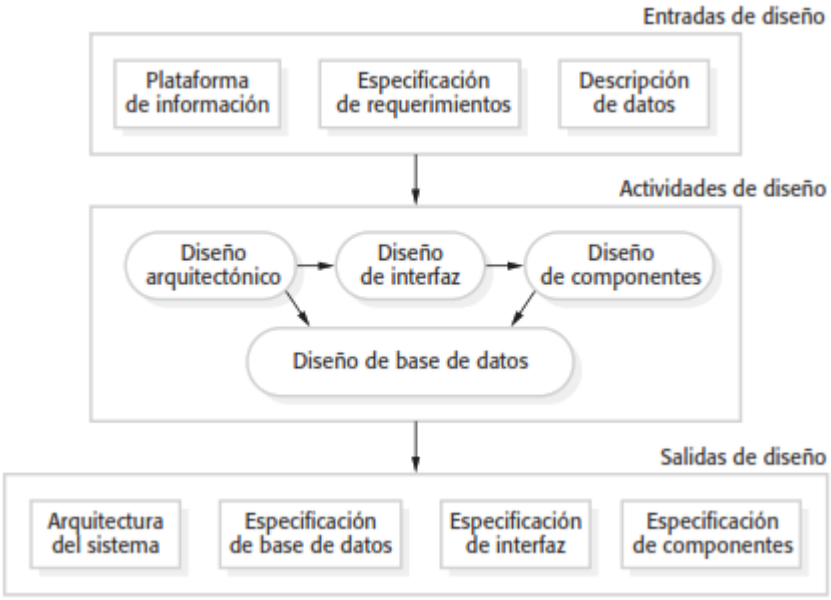
i) Los requisitos de Fiabilidad.

j) Credibilidad de la aplicación.

2.2.3 Diseño e implementación del software

El diseño del software se entiende como una descripción de la estructura del software que se va a implementar, los modelos y las estructuras de datos utilizados por el sistema, las interfaces entre componentes del sistema y en ocasiones los algoritmos utilizados.

La implementación del desarrollo de software es el proceso de convertir una especiación del sistema en un sistema ejecutable, aquí se incluyen procesos de diseño y programación de software, también se puede realizar la corrección de una especificación de software ya establecida anteriormente.



1.3 Modelo general del proceso de diseño.

El diagrama sugiere que las etapas del proceso del diseño son secuenciales, y que están vinculadas entre sí, en todos los procesos de diseño es inevitable la retroalimentación de una etapa a otra, la mayoría del software tiene interfaz junto con otros sistemas de software, en ello se incluyen los sistemas operativos, bases de datos, middleware y otros sistemas de aplicación.

La especificación de requerimientos es una descripción de la funcionalidad que debe brindar el software, en conjunto con los requerimientos de rendimiento y confiabilidad;

las actividades en el proceso de diseño varían dependiendo del tipo de sistema a desarrollar, por ejemplo, los sistemas de tiempo real precisan del diseño de temporización, pero sin incluir alguna base de datos por lo que no hay que integrar un diseño de base de datos.

Entre las actividades que forman parte del proceso de diseño para sistemas de información se encuentran:

1. Diseño arquitectónico, aquí se identifica la estructura global del sistema, los principales componentes (llamados en ocasiones subsistemas o módulos), sus relaciones y cómo se distribuyen.

2. Diseño de interfaz, en éste se definen las interfaces entre los componentes de sistemas. Esta especificación de interfaz no tiene que presentar ambigüedades. Con una interfaz precisa, es factible usar un componente sin que otros tengan que saber cómo se implementó. Una vez que se acuerdan las especificaciones de interfaz, los componentes se diseñan y se desarrollan de manera concurrente.

3. Diseño de componentes, en él se toma cada componente del sistema y se diseña cómo funcionará. Esto puede ser un simple dato de la funcionalidad que se espera implementar, y al programador se le deja el diseño específico. Como alternativa, habría una lista de cambios a realizar sobre un componente que se reutiliza o sobre un modelo de diseño detallado. El modelo de diseño sirve para generar en automático una implementación.

4. Diseño de base de datos, donde se diseñan las estructuras del sistema de datos y cómo se representarán en una base de datos. De nuevo, el trabajo aquí depende de si una base de datos se reutilizará o se creará una nueva.

El diseño de un programa para implementar el sistema se sigue naturalmente de los procesos de elaboración del sistema. Aunque algunas clases de programa, como los sistemas críticos para la seguridad, por lo general se diseñan con detalle antes de comenzar cualquier implementación, es más común que se entrelacen en etapas posteriores del diseño y el desarrollo del programa.

Las herramientas de desarrollo de software se usan para Generar un programa de “esqueleto” a partir de un diseño. Esto incluye un código para definir e implementar interfaces y, en muchos casos, el desarrollador sólo necesita agregar detalles de la operación de cada componente del programa.

En general las etapas de prueba se resumen a lo siguiente:

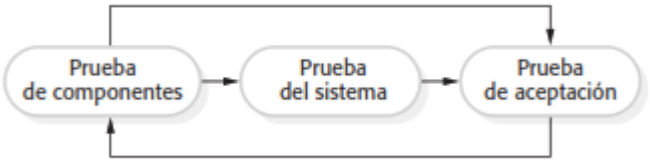


Figura 1.4 Etapas de pruebas

Por lo general los programadores realizan algunas pruebas en el código que desarrollaron, esto nos va a revelar en mayor frecuencia los defectos del programa que deben eliminarse, esta actividad se le conoce como depurar (debugging), aunque debemos diferenciar la prueba de defectos y la depuración ya que son procesos diferente, la primera únicamente establece o indica la existencia de defectos y la segunda se dedica a localizar y corregir dichos defectos.

Con la finalidad de apoyar el proceso de depuración, se deben utilizar herramientas interactivas que muestren valores intermedios de las variables del programa, así como el rastro de las instrucciones ejecutadas.

2.2.4 Pruebas y documentación del software

Hasta este punto hemos hablado de los roles que involucran a el analista, diseñador y el programador, depende del autor o conveniencia del equipo de desarrollo se pueden agrupar o desglosar las actividades, como vimos en el subcapítulo pasado la implementación es la agrupación de una parte del diseño de programas y desde luego la implementación de programas, aunque abordamos un poco de pruebas este capítulo tiene como finalidad entender y explicar la importancia de realizar un buen testing, o tener en el equipo buenos testers o “verificadores”, una forma de ver todas estas etapas de manera un poco más desglosada es la siguiente:

Figura 1.5 Roles de los desarrolladores de software (Pleeger 2002)

La validación de software o en general su verificación y validación (v&v) se crea para mostrar que un sistema cumpla tanto con sus especificaciones como con las expectativas que tenía el cliente del producto, las pruebas del programa se ejecutan con el sistema a través de datos de prueba simulados los cuales son la principal técnica de validación ,también puede incluir procesos de compromiso o validación como inspecciones y revisiones en cada etapa del proceso de software, es decir, se puede

evaluar desde el análisis y definición de requerimientos hasta el mismo desarrollo del programa o sistema.

Después o en consecuencia de la validación, el escenario ideal es que se encuentren todos los errores y finalmente el sistema se pone a prueba de los datos del cliente, sin embargo es un proceso iterativo ya que como se vayan encontrando estos se van depurando , al final los errores son componentes del sistema que saldrán a la luz durante las pruebas del sistema, por ello es conveniente tener aparte de la corrección del error una especie de retroalimentación con el fin de enriquecer la documentación del sistema.

Las etapas en el proceso de pruebas son:

1. Prueba de desarrollo

   Las personas que desarrollan el sistema ponen a prueba cada componente del mismo, tomándolos como un objeto o entidad, por lo cual usan herramientas automatizadas de pruebas como Junit que puede volver a correr pruebas de componentes cuando se crean nuevas versiones del componente.

2. Pruebas del sistema

   Es cuando los componentes del sistema se integran generando el sistema completo, esto tiene como finalidad descubrir errores que resulten de interacciones no anticipadas entre componentes y problemas de interfaz de componente, asi como demostrar que el sistema si cubre los requerimientos funcionales y no funcionales y en consecuencia poner a prueba las propiedades del sistema.

3. Pruebas de aceptación

   Esta es la etapa final en el proceso de pruebas antes de que el sistema se acepte para el uso operacional, el sistema en esta etapa ya se pone a prueba con los datos del cliente, aquí ya no se usan datos simulados, entonces se puede observar los requerimientos cumplidos y asi mismo cuando no se cumplan las necesidades del usuario.

En el siguiente diagrama podemos observar la prueba de fases en un proceso de software.

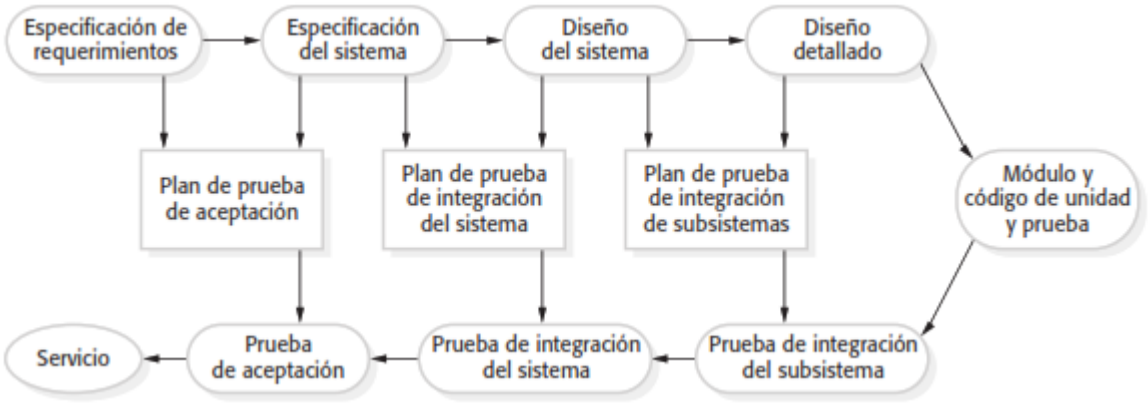


Figura 1.6 Prueba de fases en un proceso de software dirigido por un plan

2.2.5 Evolución del software (despliegue y mantenimiento)

La flexibilidad de los sistemas de software nos permite que cada vez se incorpore más a los sistemas grandes y complejos, a diferencia del hardware este una vez diseñado se complica mucho hacer cambio, pero en el software en cualquier momento durante o después del desarrollo del sistema se pueden hacer cambios, incluso aunque sean cambios grandes resultaría más fácil y menos costoso realizarlo que un cambio de hardware.

A lo largo de la historia se considera al desarrollo del software como una etapa creativa mientras que a la evolución del software lo ven como una etapa después del desarrollo, sin embargo, esta distinción entre el desarrollo y evolución es cada vez más irrelevante, ya que es mejor hacer el análisis de un sistema viendo a ambas paralelamente como un sistema continuo, donde el software cambia constantemente a los largo de su vida en función de los requerimientos y necesidades cambiantes del cliente, para lo cual nos ayudan las metodologías agiles que veremos más adelante.

2.3 Modelos de desarrollo de software

Los modelos de desarrollo de software te ofrecen un marco de trabajo que nos permite controlar el proceso de desarrollo de sistemas de información, estos marcos traen consigo una filosofía de desarrollo de programas la cual nos va a dar las herramientas necesarias para la asistencia del proceso de desarrollo.

2.3.1 Modelo en cascada

La modelo cascada toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución, luego los representa en fases separadas como especificación de requerimientos, diseño de software, implementación, pruebas y operación y mantenimiento.

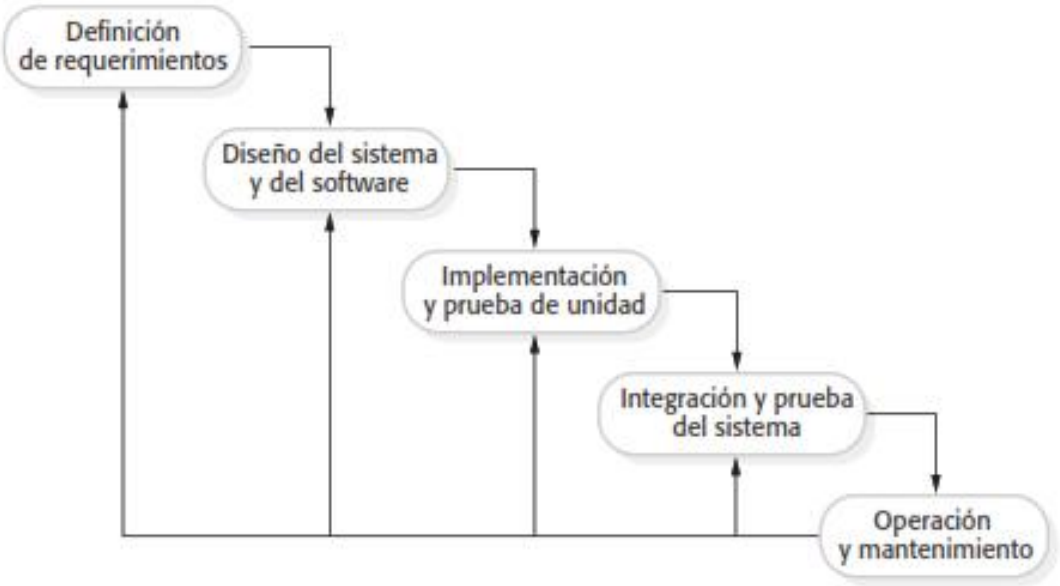


Figura 1.7 Modelo de cascada

Este proceso surge a partir de los procesos más agiles de ingeniería de sistemas, este modelo tradicional propones un proceso dirigido por un plan, primero se debe planear y programar todas las actividades del proceso, esto antes de trabajar con ellas, las principales etapas del modelo en cascada reflejan las actividades fundamentales del desarrollo las cuales describirnos en el subcapítulo anterior donde hablamos de las etapas tradicionales de desarrollo de software:

1. Análisis y desarrollo de requerimientos.
   Los servicios, restricciones y las metas del sistema se establecen mediante la consulta de usuario de sistema y después de definen con detalle y sirven como una especificación del sistema.
2. Diseño del sistema y del software.
   El proceso de diseño sistema se asigna los requerimientos para sistema de hardware o de software para establecer una arquitectura, mientras el diseño de software implica identificar y describir las abstracciones fundamentales del sistema de software y sus relaciones.
3. Implementación y prueba de unidad.
   Durante esta etapa el diseño del software pasa a ser un conjunto de programas o unidades de programa la prueba de unidad consiste en verificar que cada unidad cumpla con las especificaciones y requerimientos.
4. Integración y prueba de sistema.
   Las unidades de programa se integran y prueban como un sistema completo para asegurarse de que se cumplan los requerimientos de software, después de liberarlo y hacer algunas pruebas se libera el software al cliente.
5. Operación y mantenimiento.
   Esta suele ser una etapa larga para el ciclo de vida, aquí es donde el sistema se instala y se pone en práctica, el mantenimiento también consiste en corregir los errores que no se detectaron en etapas anteriores, esto es para mejorar la implementación de las unidades de sistema e incrementar los servicios del sistema conforme se descubren nuevos requerimientos.

Cada fas consiste en uno o más documentos que se autorizaron es decir se firmaron, y no se puede cambiar de una fase a otra sin que se termine la fase previa y esté debidamente autorizado, pero en la práctica muchas veces las etapas de traslapan y se retroalimentan mutuamente ya que no es un proceso lineal sino un proceso circular y recurrente, así que, documentos de fases anteriores se pueden modificar con la finalidad de nutrir el desarrollo de software.

Por la producción y aprobación de documentos las iteraciones suelen ser algo tedioso e implica rediseñar un diseño significativo, por ello después de un pequeño número de iteraciones es normal que se detengan o atrasen algunas partes del desarrollo, el frenado prematuro de los requerimientos significa que probablemente el sistema no hará lo que el usuario desea.

Durante la fase final del ciclo de vida, que es la operación y mantenimiento el software se pone en servicio, se descubren los errores y las omisiones de los requerimientos originales del software, y saltan a la vista los errores de programa y diseño y se detecta la necesidad de nuevas funcionalidades, por lo tanto, el sistema debe evolucionar para mantenerse vigente y permanecer útil.

El modelo de cascada es consecuente con otros modelos de proceso de ingeniería y en cada fase se produce documentación, esto hace que el proceso sea transparente y visible de modo que los administradores monitorean el progreso del plan de desarrollo, la principal problemática es la partición inflexible del proyecto en distintas etapas.

Los procesos basados en transformaciones formales se usan por lo general solo en el desarrollo de sistemas críticos para protección o seguridad, se requiere de experiencia especializada para la mayoría de los sistemas este proceso no se ofrece costo/beneficios significativos sobre otro enfoque en el desarrollo de sistema.

2.3.2 Modelo espiral

Surge por Boehm que propuso un marco de proceso de software que se dirige por el riesgo , el cual es el modelo espiral , aquí como podemos ver se representa el modelo en un espiral y no como una secuencia de actividades con cierto retraso de una actividad a otra, cada ciclo en la espiral representa una fase del proceso de software, por ende el ciclo más interno puede relacionarse con la factibilidad del sistema, el siguiente ciclo con la definición de requerimientos el ciclo que sigue con el diseño del sistema, el modelo espiral combina el evitar el cambio con la tolerancia al cambio, lo anterior supone que los cambios son resultado de riesgos del proyecto e incluye actividades de riesgo explicitas para reducir tales riesgos .

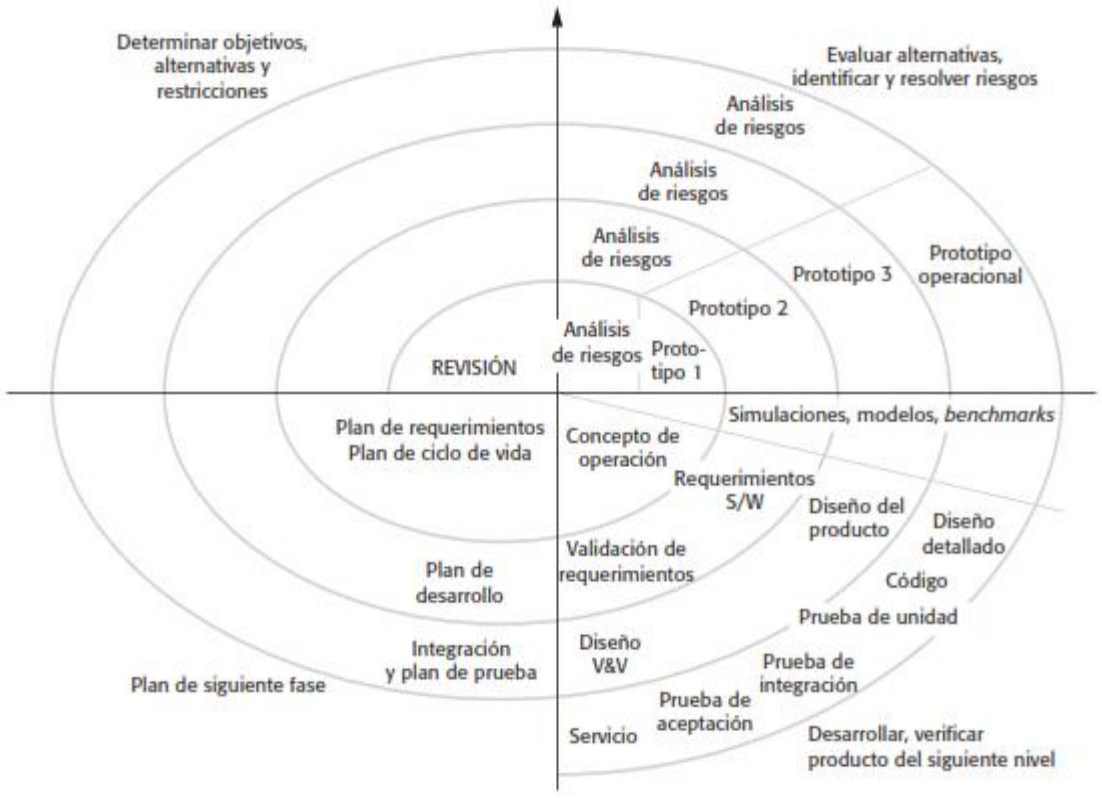


Figura 1.8 Modelo en espiral de Boehm del proceso de software (IEEE, 1988)

Cada ciclo en el espiral se divide en 4 sectores:

1. Establecimiento de objetivos

   Se definen objetivos específicos para dicha frase de proyecto, se identifican las restricciones y riesgos del proyecto, por ello se pueden identificar riesgos y replantear estrategias.
2. Valoración y reducción del riesgo

   En cada uno de los riesgos identificados del proyecto se realiza un análisis minucioso, si existe un riesgo de que los requerimientos sean inadecuados, puede desarrollarse un sistema prototipo.
3. Desarrollo y validación

   Después de una evaluación de riesgo , se elige un modelo de desarrollo para el sistema, por ejemplo los prototipos desechables sería el mejor enfoque de desarrollo, si predominan los riesgos de la interfaz del usuario por ejemplo si la principal consideración son los riesgos de seguridad el desarrollo con base en transformación formales seria el procesos más adecuado, si el principal riesgo identificado es la integración de subsistemas el modelo cascada sería el mejor modelo de desarrollo para utilizar.
4. Planeación

   El proyecto se revisa y se toma una decisión sobre si hay que continuar con otro ciclo de espiral, si se desea continuar entonces se trazan los planes para la siguiente fase del proyecto

La diferencia principal entre el modelo en espiral con otros modelos de proceso de software es su reconocimiento explícito del riesgo. Un ciclo de la espiral comienza por elaborar objetivos como rendimiento y funcionalidad. Luego, se numeran formas alternativas de alcanzar dichos objetivos y de lidiar con las restricciones en cada uno de ellos.

Cada alternativa se valora contra cada objetivo y se identifican las fuentes de riesgo del proyecto. El siguiente paso es resolver dichos riesgos, mediante actividades de recopilación de información, como análisis más detallado, creación de prototipos y simulación , después de ser valorados los riesgos se tiene que realizar cierto desarrollo seguido de una actividad de planeación para la siguiente pase del proceso, de manera coloquial el riesgo significa que algo podría salir mal, los riesgos conducen a propuestas de cambios de software y a problemas de proyecto como exceso en las fechas y el costo .

De manera simplificada este proceso de 4 etapas se ve aproximadamente así:

Figura 1.8.1 Modelo espiral resumido

2.3.3 Desarrollo iterativo e incremental

En este enfoque se vinculan las actividades de especificación, desarrollo y validación, el sistema se desarrolla como una serie de versiones (incrementos), y cada versión añade funcionalidad a la versión anterior, entonces de cierta manera cada versión es más fructífera que la anterior, también se basa en la idea de diseñar una implementación inicial, exponer esta al usuario y luego desarrollarla en sus diversas actividades hasta producir el sistema adecuado, las actividades de especificación, desarrollo y validación están entrelazadas en vez de separadas, con una retroalimentación rápida a través de las actividades.

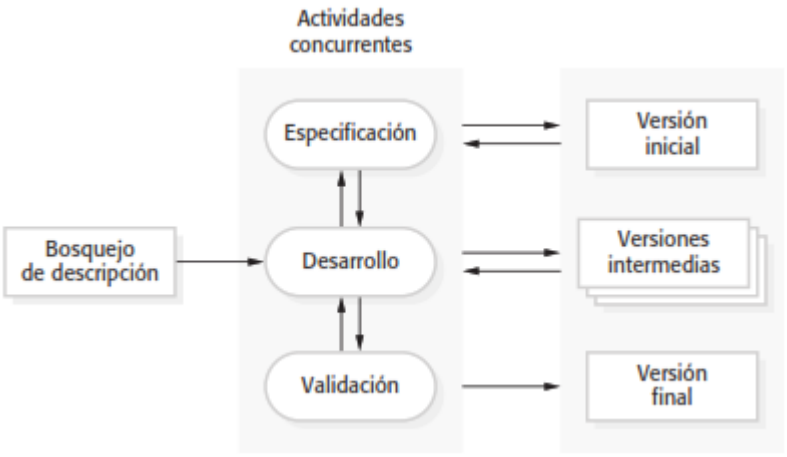


Figura 1.9 Desarrollo incremental

El desarrollo de software incremental es una parte de fundamental de los enfoques agiles, es mejor que un enfoque en cascada para la mayoría de los sistemas empresariales, de comercio electrónico y personales, este refleja la forma en la que se resuelven problemas, rara vez se trabaja por adelantado en una solución completa, al desarrollar software de manera incremental, resulta más barato y fácil realizar cambios en el software conforme este se diseña.

Cada incremento o versión nueva del sistema incorpora algunas de las funcionalidades que necesita el cliente, por lo general los primeros incrementos del sistema incluyen a función más importante o la más urgente, por ello el cliente puede evaluar el desarrollo

del sistema en una etapa relativamente temprana para constatar si se entrega lo que requiere, a diferencia del modelo de cascada el desarrollo incremental tiene tres beneficios importantes:

1. Se reduce el costo de adaptar los requerimientos cambiantes del cliente. La cantidad de análisis y la documentación que tiene que reelaborarse son mucho menores de lo requerido con el modelo en cascada.
2. Es más sencillo obtener retroalimentación del cliente sobre el trabajo de desarrollo que se realizó. Los clientes pueden comentar las demostraciones del software y darse cuenta de cuánto se ha implementado. Los clientes encuentran difícil juzgar el avance a partir de documentos de diseño de software.
3. Es posible que sea más rápida la entrega e implementación de software útil al cliente, aun si no se ha incluido toda la funcionalidad. Los clientes tienen posibilidad de usar y ganar valor del software más temprano de lo que sería posible con un proceso en cascada.

Aunque el desarrollo incremental tiene muchas ventajas, no está exento de problemas, la principal causa es la dificultad es el hecho de que las grandes organizaciones tienen procedimientos burocráticos que han evolucionado con el tiempo, y esto puede suscitar falta de coordinación entre dichos procedimientos y un proceso iterativo o ágil más informal.

El desarrollo incremental ahora es en cierta forma el más común para el desarrollo de aplicaciones, el enfoque puede estar basado en un plan, ser ágil o más usualmente las mezclas de dichos enfoques, en un enfoque basado en un plan se identifican por adelantado los incrementos del sistema, si se adopta un enfoque ágil se detectan los primeros incrementos, desde una perspectiva administrativa el enfoque incremental tiene dos problemas los cuales son:

1. El proceso no es visible. Los administradores necesitan entregas regulares para medir el avance. Si los sistemas se desarrollan rápidamente, resulta poco efectivo en términos de costos producir documentos que reflejen cada versión del sistema.
2. La estructura del sistema tiende a degradarse conforme se tienen nuevos incrementos. A menos que se gaste tiempo y dinero en la refactorización para mejorar el software, el cambio regular tiende a corromper su estructura. La incorporación de más cambios de software se vuelve cada vez más difícil y costosa.

Como vemos los problemas del desarrollo incremental se tornan difíciles principalmente para sistemas grandes, complejos y de larga duración donde diversos equipos desarrollan diferentes partes del sistema , los grandes sistemas necesitan de un marco o arquitectura estable y es necesario definir con claridad respecto a la arquitectura, esto debe planearse con anticipación en vez de desarrollarse de manera incremental, se puede desarrollar un sistema incremental y exponerlo a los clientes para su comentario, sin entregarlo o implementarlo en el entorno del cliente, las entregas y las implementaciones incrementales significan que el software se usa en procesos operacionales reales, esto no siempre es posible ya que a muchas empresas se les complica alterar su procesos normales.

2.3.4 Modelo basado en la reutilización.

Teniendo una forma iterativa e incremental suele hacerse mucho uso de la reutilización de código, sucede esto de manera informal cuando las personas trabajan en el proyecto conocen diseños o códigos que son similares a los que se requiere, los buscan o modifican según se necesite y los incorporan en sus sistemas esta reutilización ocurre independientemente del procesos de desarrollo que se emplee sin embargo en este siglo los procesos de desarrollo de software que se enfocaban en la reutilización de software existentes, los enfoques orientados a la reutilización se apoyan en componentes de software reutilizables y en la integración de marcos para la composición de dichos componentes, un ejemplo grafico de este modelo basado en la reutilización:

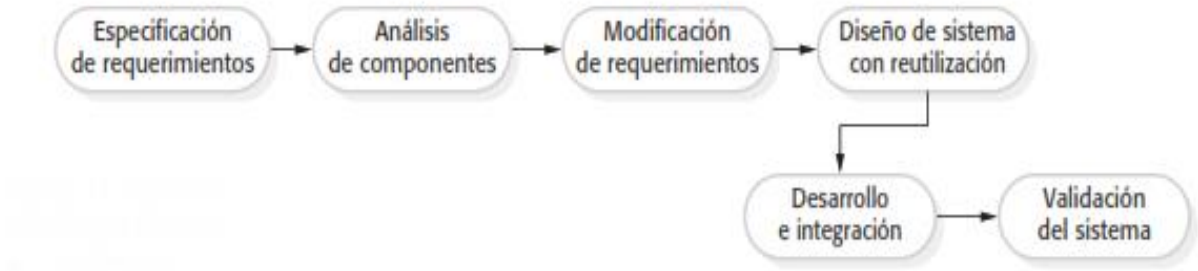


Figura 2.0 Ingeniería de software orientada a la reutilización.

Aunque la etapa inicial de especificación de requerimientos y la etapa de validación se comparan con otros procesos de software en un proceso orientado a la reutilización, las etapas intermedias son diferentes.

Dichas etapas son:

1. Análisis de componentes:

   Dada la especificación de requerimientos, se realiza una búsqueda de componentes para implementar dicha especificación. Por lo general, no hay coincidencia exacta y los componentes que se usan proporcionan sólo parte de la funcionalidad requerida.

2. Modificación de requerimientos:

   Durante esta etapa se analizan los requerimientos usando información de los componentes descubiertos. Luego se modifican para reflejar los componentes disponibles. Donde las modificaciones son imposibles, puede regresarse a la actividad de análisis de componentes para buscar soluciones alternativas.

3. Diseño de sistema con reutilización:

   Durante esta fase se diseña el marco conceptual del sistema o se reutiliza un marco conceptual existente. Los creadores toman en cuenta los componentes que se reutilizan y organizan el marco de referencia para atenderlo. Es posible que deba diseñarse algo de software nuevo, si no están disponibles los componentes reutilizables.

4. Desarrollo e integración:

   Se diseña el software que no puede procurarse de manera externa, y se integran los componentes y los sistemas COTS para crear el nuevo sistema. La integración del sistema, en este modelo, puede ser parte del proceso de desarrollo, en vez de una actividad independiente.

Existen tres tipos de componentes de software que pueden usarse en un proceso orientado a la reutilización:

1. Servicios Web que se desarrollan en concordancia para atender servicios estándares y que están disponibles para la invocación remota.

2. Colecciones de objetos que se desarrollan como un paquete para su integración con un marco de componentes como .NET o J2EE.

3. Sistemas de software independientes que se configuran para usar en un entorno particular.

Este modelo tiene la ventaja de reducir la cantidad de software a desarrollar y por lo tanto disminuir costos y riesgos, por el general también conduce a entregas más rápidas de software.

2.3.5 Proceso unificado racional (RUP)

El proceso unificado racional (RUP) es un ejemplo del proceso moderno y se derivó del trabajo sobre UML y el proceso asociado de desarrollo de software unificado, también llamado modelo de proceso hibrido, conjunta todos los elementos de los modelos de procesos genéricos e ilustra la buena práctica de especificaciones y diseño y apoya la creación de prototipos y entrega incremental.

El RUP reconoce que los modelos de proceso convencionales presentan una sola visión del proceso, el RUP por lo general se describe desde 3 perspectivas:

1. Una perspectiva dinámica que muestra las fases del modelo a través del tiempo.
2. Una perspectiva estática que presenta las actividades del proceso que se establecen.
3. Una perspectiva práctica que sugiere buenas prácticas a usar durante el proceso.

La mayoría de las descripciones del RUP buscan combinar las perspectivas estática y dinámica en un solo diagrama, esto hace que el proceso resulte más difícil de entender por lo que en este texto se usan descripciones separadas de cada una de las perspectivas, el RUP es un modelo en fases que identifica 4 fases discretas en el proceso de software, sin embargo, a diferencia del modelo cascada donde las fases se igualan con actividades del proceso, las fases en el RUP están más estrechamente vinculadas con la empresa que con las preocupaciones técnicas.

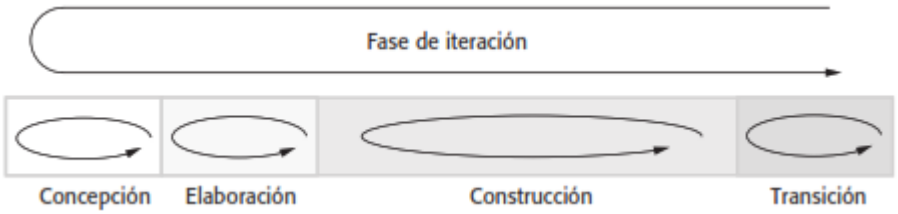


Figura 2.1 Fases en el Proceso Unificado Racional

Las etapas consisten en:

1. Concepción:

   La meta de la fase de concepción es establecer un caso empresarial para el sistema. Deben identificarse todas las entidades externas (personas y sistemas) que interactuarán con el sistema y definirán dichas interacciones. Luego se usa esta información para valorar la aportación del sistema hacia la empresa.

2. Elaboración:

   Las metas de la fase de elaboración consisten en desarrollar la comprensión del problema de dominio, establecer un marco conceptual arquitectónico para el sistema, diseñar el plan del proyecto e identificar los riesgos clave del proyecto. Al completar esta fase, debe tenerse un modelo de requerimientos para el sistema, que podría ser una serie de casos de uso del UML, una descripción arquitectónica y un plan de desarrollo para el software.

3. Construcción:

   La fase de construcción incluye diseño, programación y pruebas del sistema. Partes del sistema se desarrollan en paralelo y se integran durante esta fase. Al completar ésta, debe tenerse un sistema de software funcionando y la documentación relacionada y lista para entregarse al usuario.

4. Transición

   La fase final del RUP se interesa por el cambio del sistema desde la comunidad de desarrollo hacia la comunidad de usuarios, y por ponerlo a funcionar en un ambiente real. Esto es algo ignorado en la mayoría de los modelos de proceso de software, aunque, en efecto, es una actividad costosa y en ocasiones problemática. En el complemento de esta fase se debe tener un sistema de software documentado que funcione correctamente en su entorno operacional.

La iteración con el RUP se apoya en dos cosas cada fase puede representarse en una forma iterativa con los resultados desarrollados incrementalmente , la visión estática del RUP se enfoca en actividades que tienen más lugar durante el proceso de desarrollo se les llama flujos de trabajo, las ventajas que presentar las visiones dinámica y estática radica en que las fases del proceso de desarrollo no están asociadas con un flujo de trabajo específico, en las fases iniciales del proceso es probable que se use mayor esfuerzo en los flujos de trabajo como modelado del negocio y requerimientos y, en fases posteriores en las pruebas y el despliegue.

Los flujos de trabajo estáticos en RUP son:

Flujo de trabajo	Descripción
Modelado del negocio	Se modelan los procesos de negocios utilizando casos de uso de la empresa.
Requerimientos	Se identifican los actores que interactúan con el sistema y se desarrollan casos de uso para modelar los requerimientos del sistema.
Análisis y diseño	Se crea y documenta un modelo de diseño utilizando modelos arquitectónicos, de componentes, de objetos y de secuencias.
Implementación	Se implementan y estructuran los componentes del sistema en subsistemas de implementación. La generación automática de código a partir de modelos de diseño ayuda a acelerar este proceso.
Pruebas	Las pruebas son un proceso iterativo que se realiza en conjunto con la implementación. Las pruebas del sistema siguen al completar la implementación.
Despliegue	Se crea la liberación de un producto, se distribuye a los usuarios y se instala en su lugar de trabajo.
Administración de la configuración y del cambio	Este flujo de trabajo de apoyo gestiona los cambios al sistema (véase el capítulo 25).
Administración del proyecto	Este flujo de trabajo de apoyo gestiona el desarrollo del sistema (véase los capítulos 22 y 23).
Entorno	Este flujo de trabajo pone a disposición del equipo de desarrollo de software, las herramientas adecuadas de software.

Figura 2.2 Flujos de trabajo estáticos en el RUP.

El enfoque práctico del RUP describe las buenas prácticas de ingeniería de software que se recomiendan para su uso en el desarrollo de sistemas. Las seis mejores prácticas fundamentales que se recomiendan son:

1. Desarrollo de software de manera iterativa:
   Incrementar el plan del sistema con base en las prioridades del cliente, y desarrollar oportunamente las características del sistema de mayor prioridad en el proceso de desarrollo.

2. Gestión de requerimientos:
   Documentar de manera explícita los requerimientos del cliente y seguir la huella de los cambios a dichos requerimientos. Analizar el efecto de los cambios sobre el sistema antes de aceptarlos.

3. Usar arquitecturas basadas en componentes:
   Estructurar la arquitectura del sistema en componentes, como se estudió anteriormente en este capítulo.

4. Software modelado visualmente:
   Usar modelos UML gráficos para elaborar representaciones de software estáticas y dinámicas.

5. Verificar la calidad del software:
   Garantizar que el software cumpla con los estándares de calidad de la organización.

6. Controlar los cambios al software:
   Gestionar los cambios al software con un sistema de administración del cambio, así como con procedimientos y herramientas de administración de la configuración.

2.4 Metodologías agiles

La filosofía de las metodologías agiles consiste en darle mayor valor al individuo, a la colaboración del cliente y al desarrollo incremental del software con iteraciones cortas, este enfoque ha demostrado tener muchas efectividades en proyecto con requisito cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad.

La revolución de las metodologías agiles es evidente al momento de la producción de software en menos tiempo y a la vez generando un amplio debate entre los seguidores y quien por escepticismo o convencimiento no las ven como una alternativa para las metodologías tradicionales , este trabajo se presenta resumidamente el contexto en el que surgen las metodologías agiles, sus valores, principios y comparación con metodologías tradicionales, además de describir brevemente las principales propuestas especialmente programación extrema (la cual veremos más adelante), como sabemos estas están más orientadas o han sido mejor adaptadas para proyectos pequeños, las metodologías agiles construyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las practicas esenciales para asegurar la calidad del producto.

En febrero del 2001 tras una reunión nació el termino ágil aplicado a el desarrollo del software, su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software de manera rápida y respondiendo a los cambios que puedan surgir a lo largo del proyecto.

Tras esto se creó The Agile Alliance la cual es una organización sin fines de lucro dedicada a promover los conceptos relacionados con el desarrollo de software ágil de software y ayudar a las organizaciones para que adopten dichos conceptos, siendo el punto de partida el manifiesto por el desarrollo ágil de software.

2.4.1 SCRUM

Define un marco para gestión de proyectos, está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos:

1. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente.
2. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

Los principios de scrum son congruentes con el manifiesto ágil y se utilizan para guiar actividades de desarrollo dentro de un proceso de análisis que incorpora las actividades estructurales de requerimientos, análisis, diseño, evolución y entrega, que, como vemos son congruentes con las etapas tradicionales del desarrollo de software que vimos en capitulo anteriores, en cuanto a los principios del manifiesto de desarrollo ágil destacan:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Scrum acentúa el uso de un conjunto de patrones de proceso de software que han demostrado ser eficaces para proyectos con plazos de entrega muy apretados, requerimientos cambiantes y negocios críticos. Cada uno de estos patrones de proceso define un grupo de acciones de desarrollo:

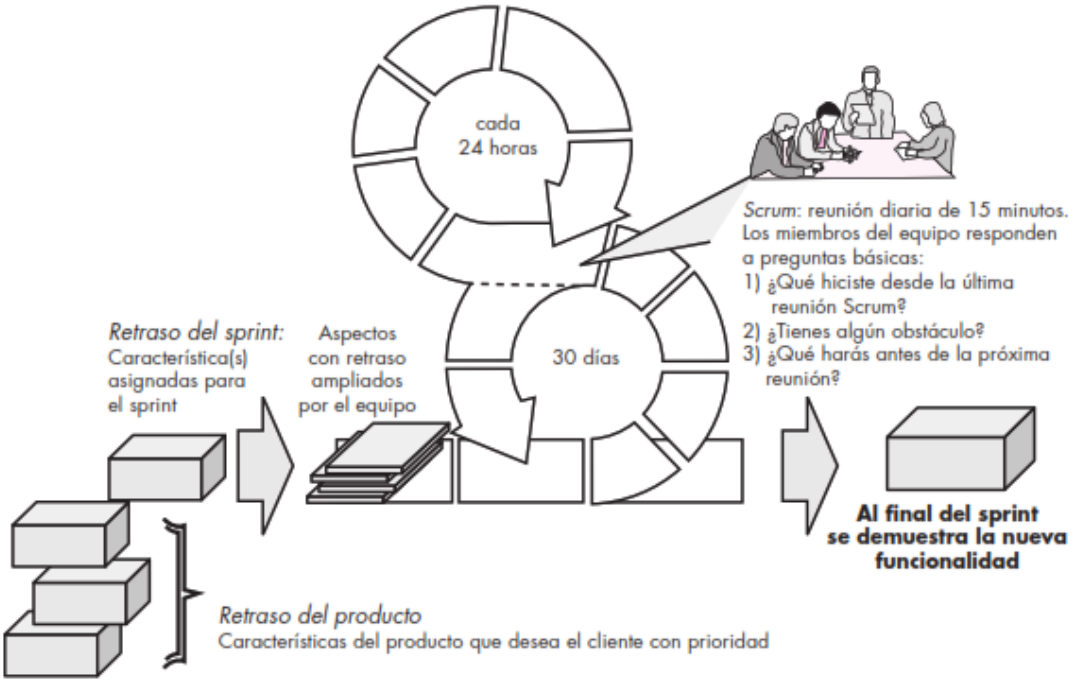


Figura 2.3 Flujo de scrum.

Retraso: Es la lista de prioridades de los requerimientos o necesidades del proyecto que dan al cliente un valor del negocio, lógicamente se pueden agregar más aspectos al retraso conforme sean necesarios, el gerente del proyecto evalúa el retraso y actualiza las prioridades según se requiera.

Sprints: Consiste en tiempos de trabajo que se necesitan para cumplir algún requerimiento definido en el retraso que debe ajustarse a una caja de tiempo (tiempo indicado para cumplir alguna tarea) predefinida, durante el sprint no se introducen cambios, así el sprint permite a los miembros del equipo trabajar en un ambiente de corto plazo, pero estable.

Reuniones Scrum: Son reuniones que tienen una duración de 15 minutos por lo general, y estas son diario, mientras se realiza la reunión todos los integrantes del equipo deben de responder las siguientes preguntas:

- ¿Qué hiciste desde la última reunión del equipo?
- ¿Qué obstáculos estás encontrando?
- ¿Qué planeas hacer mientras llega la siguiente reunión del equipo?

Maestro Scrum o Scrum master: Es un líder del equipo que dirige la junta y evalúa las respuestas de cada persona, la junta scrum ayuda al equipo a descubrir problemáticas, así mismo en las juntas se da la “socialización del conocimiento” (lo cual ayuda a solucionar problemáticas en equipo).

Demostraciones preliminares: Es la entrega del incremento de software al cliente de modo que las funcionalidades que se hayan implementado pueda verlas el cliente y este las pueda evaluar para determinar si cumplen o no con los requerimientos, es importante señalar que no es necesario entregar toda la funcionalidad, solo se entrega lo que se establece en la caja de tiempo.

2.4.2 Extreme Programming (XP)

Programación extrema (XP) es una metodología ágil de desarrollo de software centrada en potenciar las relaciones interpersonales como clave para el éxito de desarrollo de software, focalizando en el trabajo en equipo, impulsando el aprendizaje colectivo entre desarrolladores, e incentivando un clima de trabajo agradable, parte del nombre se debe a que practicas con sentido común las llevan al extremo, cabe destacar que utilizaremos definiciones y principio de Kent Beck, el autor original de la metodología.

XP se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación efectiva ente los involucrados, asi como simplicidad en las soluciones implementadas y una capacidad alta para adaptarse a los cambios, XP funciona de mejor manera en proyectos con requerimientos imprecisos y muy cambiantes y donde existe un alto riesgo técnico.

Historias de Usuario

Son una técnica utilizada para especificar los requisitos del software, y son prácticamente tarjetas de papel en las cuales el cliente enlista y describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales, la forma en la que se manejan estas tarjetas es muy dinámico y flexible, cada historia debe ser lo suficientemente comprensibles y delimitadas para que los programadores lo puedan implementar en unas semanas.

Dichas tarjetas deben tener aspectos como fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, numero de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimulación técnica, descripción, notas y una lista de seguimiento con la fecha, estado, cosas por terminar y comentarios, una historia puede realizarse de una a tres semanas de tiempo de programación.

CUSTOMER STORY CARD

Story Card No: System:
Story Card Name: Date:
Customer: Customer Effort writing SC (hrs):

Type of activity (New Story/Fix Defect/Enhance New Feature):

Failure Risk Impact (High/Medium/Low): Priority (High/Medium/Low):

Acceptance 'Functional' Test Scenario No:

Story Description

Pre-conditions
1.
…
Story Description

…
Post-conditions
1.
…
Notes:

Figura 2.4 Customer story and task card.

Roles de XP:

Programador: El programador escribe las pruebas unitarias y produce el código del sistema.

Cliente:

Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

Encargado de pruebas (Tester):

Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para prueba.

Encargado de seguimiento (Tracker):
Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.

Entrenador (Coach):
Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

Consultor:
Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

Gestor (Big boss):
Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

El proceso de XP a grandes rasgos consiste de:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En las iteraciones de este ciclo aprende tanto el cliente como el programador, no debe existir una presión al programador a realizar más trabajo que el estimado, ya que se perderá la calidad en el software o simplemente no se cumplirán los plazos, de la

misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible en cada iteración.

El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

Prácticas de XP:

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione, además de hacer uso de las siguientes practicas:

- El juego de la planificación:
  Hay una comunicación frecuente el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.

- Entregas pequeñas:
  Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.

- Metáfora.
  El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).

- Diseño simple:
  Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

- Pruebas:
  La producción de código está dirigida por las pruebas unitarias. Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.

- Refactorización (Refactoring):
  Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.

- Programación en parejas:
  Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores, …).

- Propiedad colectiva del código:
  Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

- Integración continua:
  Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.

- 40 horas por semana:

  Se debe trabaja r un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.

- Cliente:

  El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la comunicación escrita.

- Estándares de programación. XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

El mayor beneficio de las practicas se consigue una aplicación conjunta y equilibrada, ya que se conjuntan una con otra y pueden recibir retroalimentación sin importar la “jerarquía”, normalmente cuando una práctica retroalimenta a otra y viceversa se representa con dos líneas representando bidireccionalidad, aunque no son practicas nuevas si son prácticas que tienen como merito la efectividad y la complementación.

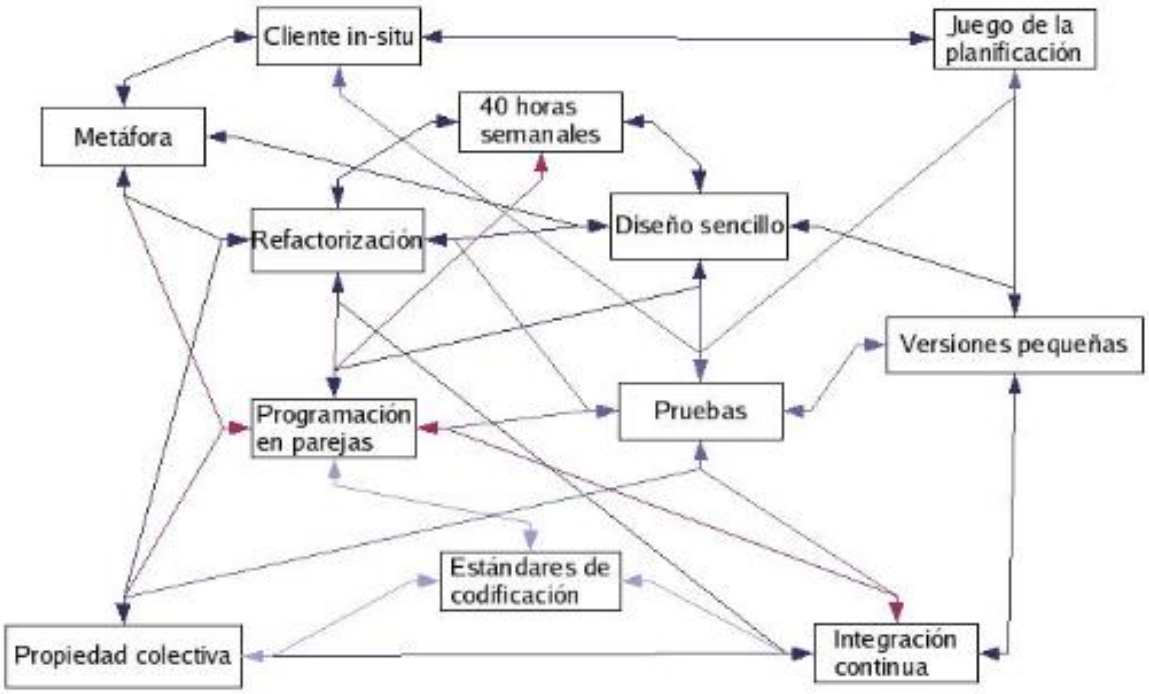


Figura 2.5 Las practicas se refuerzan entre si (ISSI 2003).

2.4.3 Lean software development (LD)

Surgió de la experiencia de Bob Charette's en proyectos con la industria japonesa del automóvil en los 80's y utilizada en proyectos de telecomunicaciones, en LD los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejores la productividad del cliente.

Las ideas fundamentales de Lean en general son

- Fabricar únicamente lo que se necesita
- Eliminar aquello que no añada valor al producto
- Detener la producción si algo va mal

Principios Lean Software

• Eliminar el desperdicio

Hacer desaparecer del proceso y el producto todo aquello que no aporta valor al cliente.

• Calidad integrada

El desarrollo ha de realizarse desde el primer momento con calidad. Las acciones correctivas han de emprenderse lo más próximo a que se detecta su necesidad y lo que es más importante, debe existir un enfoque preventivo: se deben buscar las condiciones que eviten si quiera la posibilidad de que se den errores.

• Crear conocimiento

El desarrollo de Software es un proceso de creación de conocimiento que va evolucionando a medida que se va produciendo y que, por tanto, se ha de evitar el derroche de tratar de capturarlo prematuramente. Se han de centrar esfuerzos en mejorar este conocimiento, en hacerlo más profundo y en dar respuesta al cambio.

• Aplazar las decisiones

Decidir tan tarde como sea posible: dada la frecuente incertidumbre que rodea la toma de requisitos, lo más aconsejable es retrasar las decisiones tratando de tomarlas con la mayor cantidad de información posible, y siempre adoptando una aptitud previsora ante la certeza del cambio.

• Entregar tan rápido como sea posible

Consecuencia de lo anterior, es necesario disponer de medios que permitan, una vez tomada una decisión, materializarla, sin sacrificar la calidad.

• Respetar a las personas

Reconociendo que todos queremos trabajar en productos de éxito. Desarrollando buenos líderes capaces de motivar a los equipos de trabajo, practicando y transmitiendo respeto y consideración por los demás. Dotando a las personas del conocimiento necesario para cumplir sus objetivos, estableciendo metas razonables, que puedan alcanzarse y que permitan a las personas autoorganizarse para conseguirlas.

• Optimizar el conjunto

Se debe evitar la tendencia a realizar mejoras locales a favor de un enfoque global.

Desperdicios de LD:

1. Trabajo realizado parcialmente: uno de los desperdicios del lean softwares más frecuentes.
- Documentación que tardamos meses en elaborar pero que queda sin codificar.
- Los requisitos e historias de usuario obsoletas.
- Código no probado o que responde a funcionalidad no necesaria.

2. Característica extra: aquello que creemos que el cliente va a necesitar pero que cliente no ha pedido.

- Incluir cosas en el producto que al final no se usan.
- Funcionalidad que ha quedado obsoleta.
- Características introducidas para probar la última moda y esa tecnología moderna.

3. Reaprendizaje:

- Resolver un problema y no implementarlo rápidamente
- Tiempo pasado en descubrir cosas que otro puede contarnos
- Desarrolladores reasignados
- Código mal escrito o indocumentado que conduce a una incalculable cantidad de reaprendizaje

4. De mano en mano:

- Documentos de análisis de requisitos, que luego pasan a las manos de un diseñador. El diseñador elabora un diseño y entonces pasa a manos de los Programadores.
- Poco conocimiento puede transmitirse solo con papel

5. Las pausas:

- Empezar a trabajar en el desarrollo de un proyecto mucho tiempo después del contacto inicial con el cliente.
- Esperar a que el equipo esté disponible para empezar a trabajar.
- Largas fases de documentación de requisitos.
- Revisión o aprobación de procesos que requieren de un individuo apenas disponible.

6. Cambio de tarea: el coste de cambiar de tarea durante el desarrollo de software ha sido un problema de siempre.

- Concentrarse no es fácil. Hay estudios que indican que necesitamos por lo menos quince minutos para concentrarnos en algo. Una vez concentrado, cualquier interrupción nos obliga a empezar de nuevo.

7. Defectos: uno de los desperdicios más obvios.

- Todo aquello que no se hace bien, es un desperdicio, no aporta valor, y consume tiempo a la hora de repararlo.

Figura 2.6 7 Principios de lean software development.

2.5 Stakeholders

El termino fue adaptado por primera vez por R. Edward Freeman en 1984, el cual fue originalmente adaptado a las organizaciones, pero actualmente ha sido utilizado en distintos ámbitos, lógicamente entre ellos el desarrollo de software, pero entonces, ¿Qué es un Stakeholder?, básicamente se refiere a los interesados en el proyecto, y aunque pareciera ser un aspecto poco importante en la gestión de un proyecto, es una de las partes esenciales de ingeniería de requerimientos, esto para evitar problemas en un futuro, por ello es importante identificar a los involucrados de ambas partes tanto de la parte proveedora del servicio como del cliente, esto para evitar ruido en la comunicación, en ese sentido se pueden clasificar para entender mejor como identificarlos.

2.5.1 Clasificación Sutcliffe

En este caso la clasificación de Sutcliffe entender a los stakeholders de la siguiente manera:
Stakeholders primarios: son aquellos usuarios que tendrán la tarea de operar el sistema (los operadores normales, y posiblemente también los operadores de mantenimiento).

Stakeholders secundarios: son los usuarios que no operan el sistema directamente, pero si se ven influenciados por el mismo ya que sus trabajos se verán afectados directamente por el mismo ya que el éxito de su labor dependerá del mismo también.

Stakeholders terciarios: son aquellos que no suelen trabajar directamente con el sistema, pero si hacen uso de la información que este genera para la planificación, control o toma decisiones de la empresa. Estos suelen ser los altos directivos que, si bien no se ven involucrados directamente con el sistema, también necesitan del mismo para una mejor labor.

2.5.2 Clasificación PMBOK

Sin embargo, existen otros métodos de clasificación, uno de ellos es el que hay en el PM Book el cual los clasifica en:
Stakeholder ajeno: Se caracteriza por su desconocimiento del proyecto y su potencial impacto sobre el mismo. Por ejemplo, que el líder técnico o los mismos desarrolladores no tengan noción de que el desarrollo tiene el 70% de la facturación de la empresa. El no saberlo, sumado a una falta de interés en interiorizarse por parte de estas personas podría generar que una demora o el
mismo fracaso del sistema puede llegar a ser desastroso para las finanzas de la compañía.

Stakeholder resistente: Son aquellos conscientes del proyecto y de su potencial impacto, pero que se mantienen resistentes al cambio. Por ejemplo, que el líder técnico que propuso una tecnología distinta para el desarrollo del sistema, y no esté de acuerdo con lo que finalmente se resolvió. Esto es muy común dentro de la ingeniería web, por la cantidad de lenguajes, tecnologías y técnicas existentes en la actualidad.

Stakeholder neutral: Aquellos que conocen el proyecto, pero se manifiestan imparcial sobre el mismo. El ejemplo más común para estos, son los desarrolladores que conocen su trabajo, y la empresa, y sin bien han trabajado distintos proyectos, su trabajo no suele cambiar demasiado más allá de los distintos proyectos. Por lo tanto, no suelen generar un impacto positivo o negativo en el proyecto. Esa imparcialidad que mantienen sobre los proyectos suele valorarse al no representar una amenaza para el éxito del proyecto.

Stakeholder de respaldo: Para este caso, son aquellos que conocen el proyecto, su potencial impacto y alineados con el mismo apoyan estos cambios. Un ejemplo claro para estos casos, es el de un líder de proyecto que está motivado por la firma del contrato y por la experiencia que va a adquirir trabajando en el proyecto. Estas

personas suelen hablar continuamente con todos los participantes del proyecto de los aspectos positivos del desarrollo, motivando al resto del grupo y generando un respaldo y aceptación generalizado.

Stakeholder líder: Además de ser consiente del proyecto y de su potencial impacto, colabora activamente para alcanzar el éxito del mismo. Es lo que otros autores denominan sponsors de los distintos proyectos. Como ejemplo podemos citar el de un gerente de sistemas que se siente total responsable del proyecto y ve como un gran desafío el éxito del mismo. De esta forma, se encarga de alinear los intereses de todos los participantes trabajando muy cerca de todos los involucrados, identificando y convenciendo a aquellos individuos que no demuestran interés en el proyecto, y preocupándose de que el equipo de personas cuente con todos los recursos y herramientas necesarias.

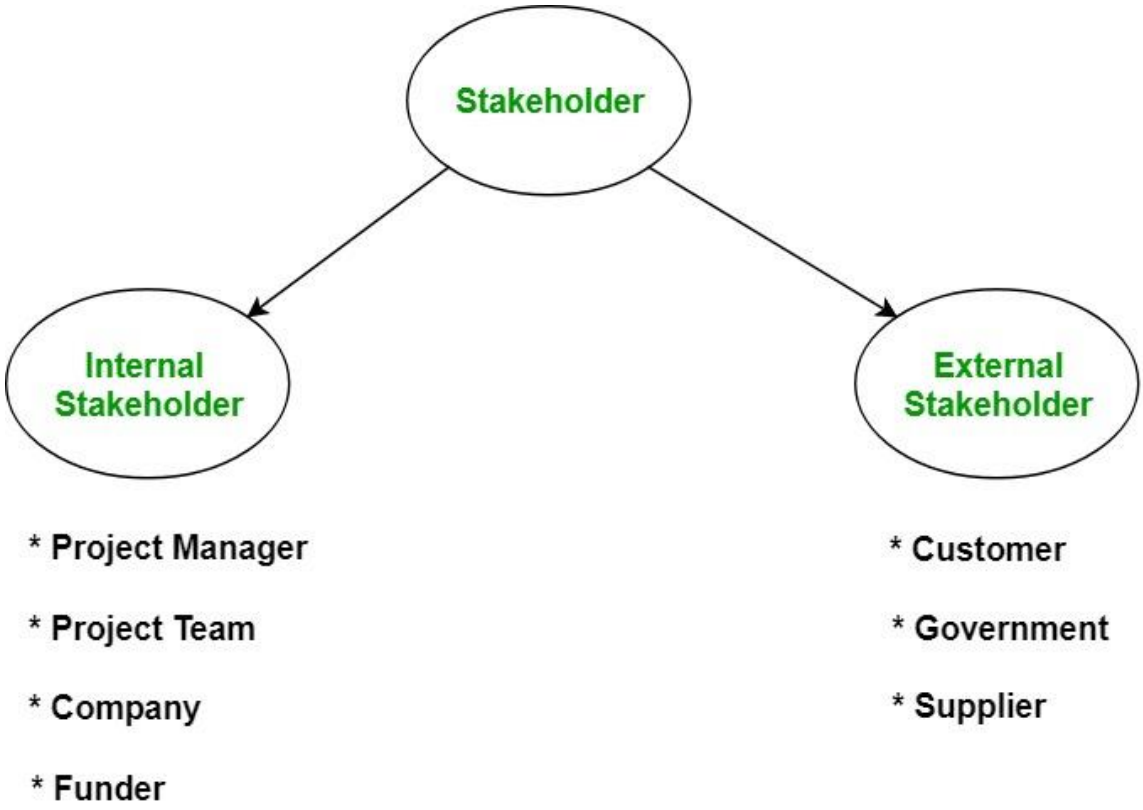


Figura 2.7 Stakeholders en ambas partes.

Capítulo III:

Caso practico

El momento de presentar mi caso práctico ha llegado, desde luego este trabajo de tesis debes de saber que tiene como objetivo recopilar en un documento tópicos importantes como : cosas que se deben hacer, cosas que no se deben hacer, buenas prácticas, malas prácticas, casos de éxito y casos de fracaso, con un enfoque en un proyecto desarrollado por estudiantes de séptimo semestre de la licenciatura en informática, previo a ello me parece razonable plantear el contexto del desarrollo del proyecto, entonces empezaremos por las necesidades o problemática a resolver.

Problemática:
En la FES Cuautitlán, se tenía la necesidad de crear contenido multimedia para presentaciones a un público predeterminado, estas normalmente empleadas por docentes, este contenido iba desde presentaciones tipo PPTX a videos interactivos con algún material en específico, y para ello se tenía que pagar un licenciamiento que muchas veces no era viable por la usabilidad.

Propuesta de solución:
Crear un sistema web basado en herramientas canvas en el cual se pueda importar contenido multimedia con la finalidad de crear presentaciones animadas o videos, y que el usuario finalmente los pueda exportar para reproducirlos, esto con una solución web que esté disponible donde sea.

Equipo de desarrollo: Inicialmente se disponía de dos recursos para desarrollar el proyecto con un dominio bajo-intermedio de tecnologías web.

Notas:

Un primer error por parte del equipo de desarrollo fue empezar a tomar con claridad un levantamiento de requerimientos, y convertirlos a especificaciones, no existieron especificaciones claras, lo cual nos ocasiono problemas.

3.1 Tecnologías empleadas.

Por las limitantes que teníamos en cuanto a presupuesto, recursos y conocimiento, decidimos utilizar el stack de tecnologías web más popular las cuales son:

HTML5:

Utilizamos esta tecnología para crear la estructura de la página web, utilizando la versión más reciente, cabe destacar que se tomó la decisión de utilizarlo por la popularidad y la amplia variedad de contenido que tiene.

Figura 2.8 HTML5

CSS3

Utilizamos esta tecnología para el desarrollo visual del frontend, y para crear y modificar recursos multimedia de el mismo sistema, además de en maquetar el proyecto.

Figura 2.9 CSS3

BOOTSTRAP 4

Bootstrap es un framework de frontend, el cual usamos para el desarrollo de UI del sistema y para la creación y gestión de recursos multimedia.

Figura 2.8 Bootstrap 4

PHP 7.3.0

PHP nos ayudó a hacer el desarrollo backend del sistema, cabe destacar que es un lenguaje muy popular del lado del servidor, es por ello también que nos decidimos a utilizarlo.

Figura 2.9 PHP 7.3.0

JavaScript

Usamos JavaScript puro y con librerías, en este caso JS puro nos sirvió para desarrollar toda la lógica y UX del frontend para el usuario.

Figura 3.0 JavaScript

CANVAS HTML

Aunque es un elemento incorporado de HTML, consideramos que era necesario señalar la participación de este elemento porque prácticamente fue la base del proyecto, cabe destacar que canvas permite la generación de gráficos dinámicamente por medio del scripting. Entre otras cosas, permite la renderización interpretada dinámica de gráficos 2D y mapas de bits, así como animaciones con estos gráficos.

Figura 3.1 Canvas

KONVA JS

Utilizamos este framework para la integración amplia de canvas, ya que amplía el contexto 2d al permitir la interactividad del lienzo para aplicaciones móviles y de escritorio.

Figura 3.2 Konva

CREATE JS

Es un conjunto de bibliotecas y herramientas modulares que trabajan juntas para crear contenido interactivo en tecnologías web abiertas a través de HTML5.

Figura 3.3 Create JS

WEBRTC

WebRTC es un proyecto libre y de código abierto que proporciona a los navegadores web y a las aplicaciones móviles comunicación en tiempo real a través de interfaces de programación de aplicaciones, y lo utilizamos para gestionar la parte de descarga de contenido en formatos multimedia.

Figura 3.4 WebRTC

POPCORN JS

Es una biblioteca de JS que utiliza las propiedades, métodos y eventos HTMLMediaElement nativos, los normaliza en una API y proporciona un sistema de complemento.

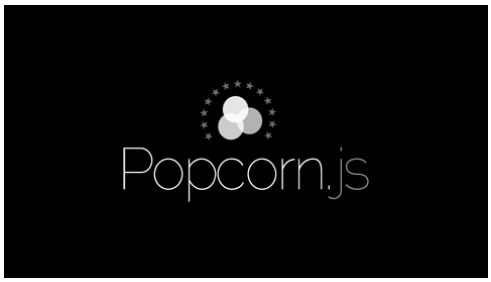


Figura 3.5 PopCorn JS

GITHUB

En este caso usamos GitHub como repositorio para trabajar en equipo.

Figura 3.6 GitHub

GIT

Utilizamos git para gestionar el control de versiones de manera local, y generar cambios más ordenados.

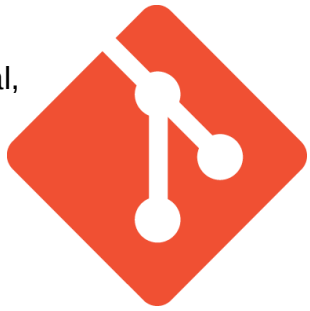


Figura 3.7 Git

3.2 Desarrollo del sistema

El proyecto originalmente tenía la intención de crearse con tecnologías puras, sin usar framework, pero después de analizar las especificaciones notamos que era una tarea que lejos de ser compleja era innecesariamente grande, por ello necesitamos refactorizar algunas partes del código, pero la estructura fue de esta manera:

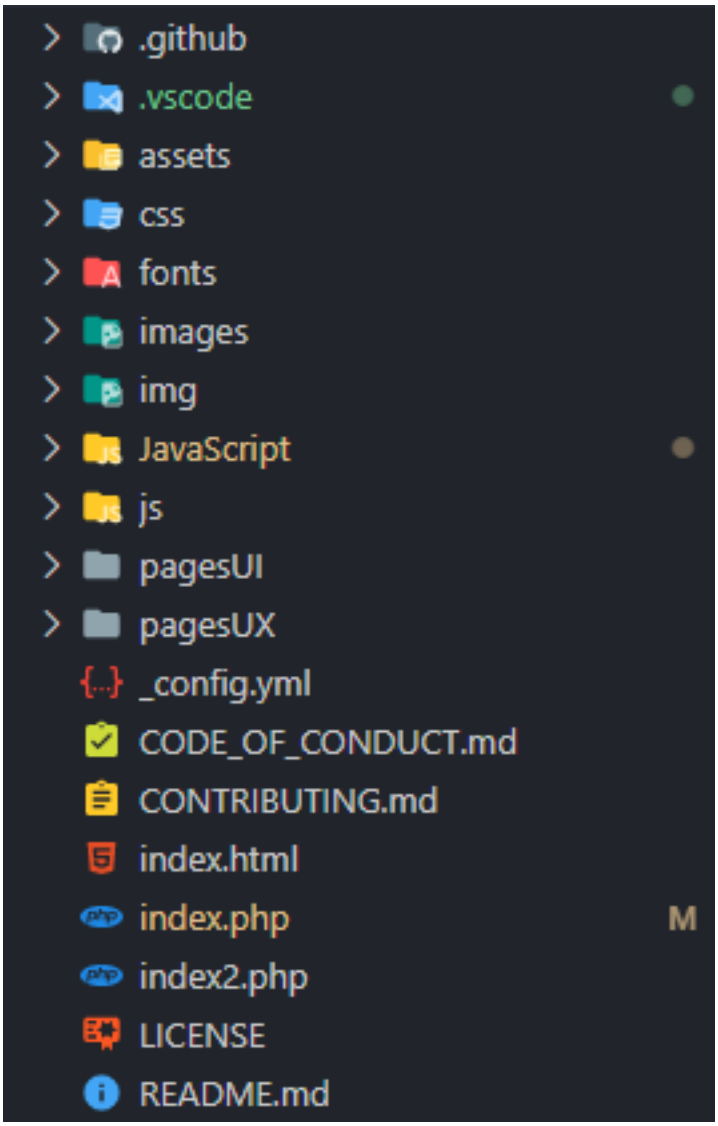


Figura 3.8 Directorio de proyecto.

- GitHub: Este directorio guarda las variables de entorno de git.
- .vscode: Este directorio contiene las variables de entorno del IDE VsCode.
- assets : Este directorio contiene los archivos css de la página de inicio la cual fue desarrollada por otro equipo.
- css: Este directorio contiene los archivos css del desarrollo de la plataforma en específico.
- fonts: Este directorio contiene las fuentes que se utilizan en el proyecto.
- Images: Este directorio contiene los recursos disponibles para el uso de la plataforma

- Img: Este directorio contiene los recursos usados en la página principal.
- JavaScript: Este directorio contiene los archivos JS del frontend de la plataforma de canvas.
- Js: Este directorio contiene los archivos JS del frontend de la página principal.
- pagesUI: Este directorio contiene los archivos de extensión para el diseño de interfaz.
- pagesUX: Este directorio contiene los archivos extensión para la experiencia del usuario.
- Archivos index: Inicializan la página principal y la plataforma.

Durante el desarrollo de la plataforma se intentó llevar un desarrollo de manera ordenada aplicando OOP, sin embargo, por las malas prácticas de algunos desarrolladores se complicó un poco la situación, de esto hablaremos en la sección de problemas, entonces como cualquier proyecto de desarrollo web iniciamos en los archivos index, y como siempre se espera en el encabezado se referencian los archivos de JS y CSS.

Como ejemplo es este fragmento de código donde referenciamos WEBRTC

```
<!-- recommended -->
<script src="https://www.WebRTC-Experiment.com/RecordRTC.js"></script>

<!-- use 5.5.6 or any other version on cdnjs -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/RecordRTC/5.5.6/RecordRTC.js"></script>
```

Figura 3.9 Redundancia de importaciones.

Como sabemos usualmente cuando usamos alguna librería o framework podemos importarla por CDN en este caso hicimos eso.
Luego tenemos por ejemplo una parte de código CSS dentro del archivo HTML, que tiene una razón de esta ahí, y esto es porque es un loader y necesitamos que sea lo primero que se cargue y en ese momento únicamente lo logramos así, entonces consideramos que era buena práctica ponerlo así, otra funcionalidad que consideramos importante fue la parte de bloquear los menús contextuales que

podemos llamar con clic derecho y F12 con la finalidad de evitar conflictos con la plataforma ya que hace uso de esos recursos, para por ejemplo cambiar las dimensiones de alguna imagen, modificar alguna parte del texto o algún cambo en general que pueda surgir, para ello también usamos un script en JS para poder lograr esta funcionalidad:

```
//BLOQUEO DE CONTEXT MENU Y F12

$(document).bind("contextmenu", function(e) {
  e.preventDefault();
});
$(document).keydown(function(e) {
  if (e.which === 123) {
    return false;
  }
}); //TERMINA BLOQUEO
(function() {
  if (!document.createElement('canvas').getContext) {
    document.body.innerHTML = '<h1 style="height:auto;color:red;font-size:30px;">Excuse me Sir,<br /><br />You are using very old
      web-browser!<br /><br />Please upgrade it.</h1>';
  }
  var prepend = function(parent, elementToPrepend) {
    return parent.insertBefore(elementToPrepend, parent.firstChild);
  };
```

Figura 4.0 Inserción código HTML en JavaScript.

Al final la página web inicial se ve así, teniendo el header con un slide de Bootstrap.

Figura 4.1 Interfaz principal de la plataforma.

También considero importante recalcar que los diseños que se muestran fueron diseñados por miembros del proyecto, y que todos ellos fueron inspirados en el uso exclusivo de la comunidad estudiantil de las FES Cuautitlán.

Luego el body en varias categorías con la finalidad de atraer la atención del usuario.

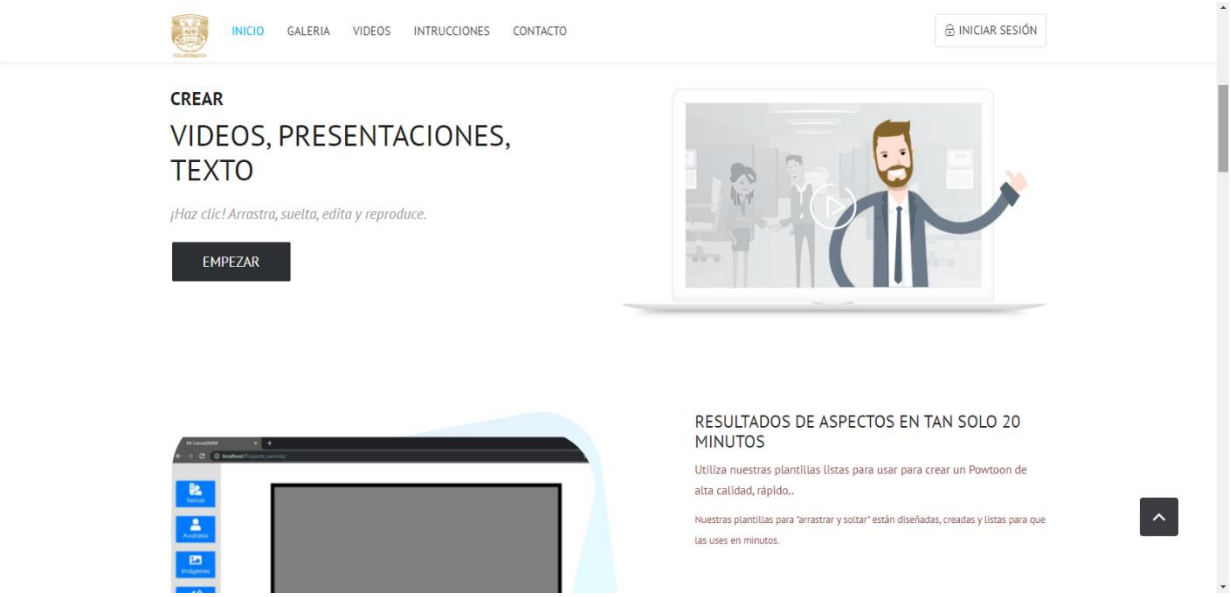


Figura 4.2 Interfaz principal de la plataforma.

También cabe destacar que la versión de la imagen es una versión algo antigua, pero de igual manera nos sirve para explicar el desarrollo del sistema; Posteriormente tenemos la categoría de diseños, que de igual manera fueron diseñados por la comunidad estudiantil.

Al desplegar el menú de personajes tenemos esto:

Figura 4.3 Galería de personajes.

Al desplegar objetos tenemos:

Figura 4.4 Galería de objetos.

Al desplegar fondos tenemos:

Figura 4.5 Galería de fondos.

Sin embargo, también tenemos contenido específicamente de la comunidad para la comunidad de FES Cuautitlan:

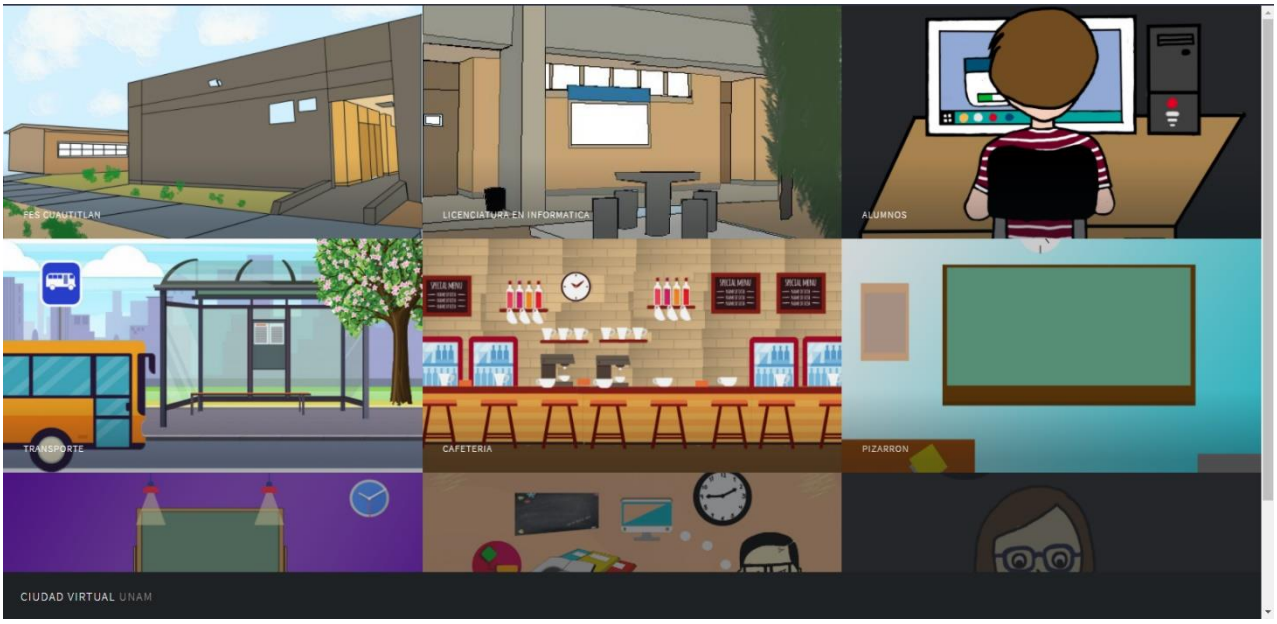


Figura 4.6 Galería de ciudad virtual.

Y esto porque el equipo considero favorable que la comunidad conozca el tipo de recursos con el que cuenta, para que hagan uso de la herramienta.

Posteriormente tenemos un apartado para que la comunidad tenga un overview de la plataforma:

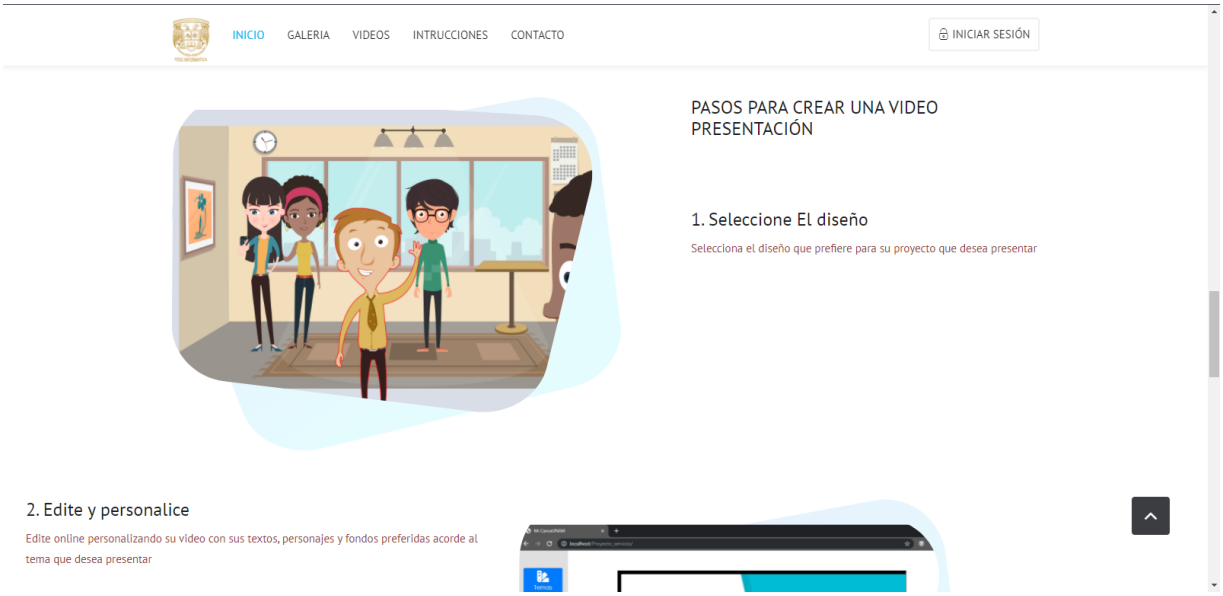


Figura 4.7 Menú de plataforma.

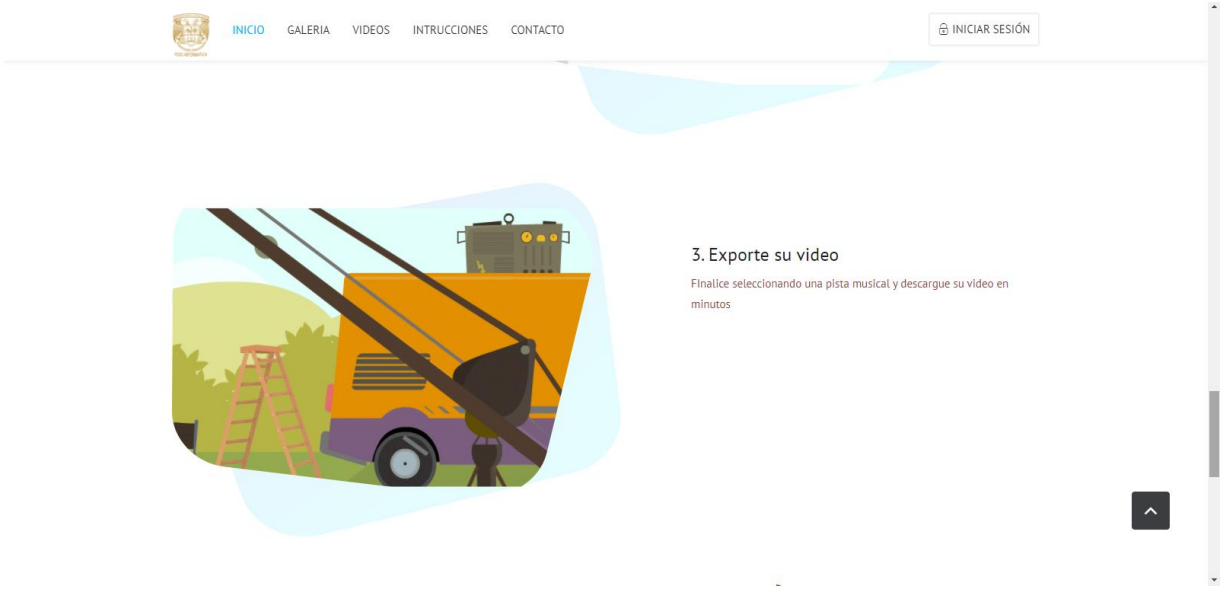


Figura 4.8 Menú de plataforma.

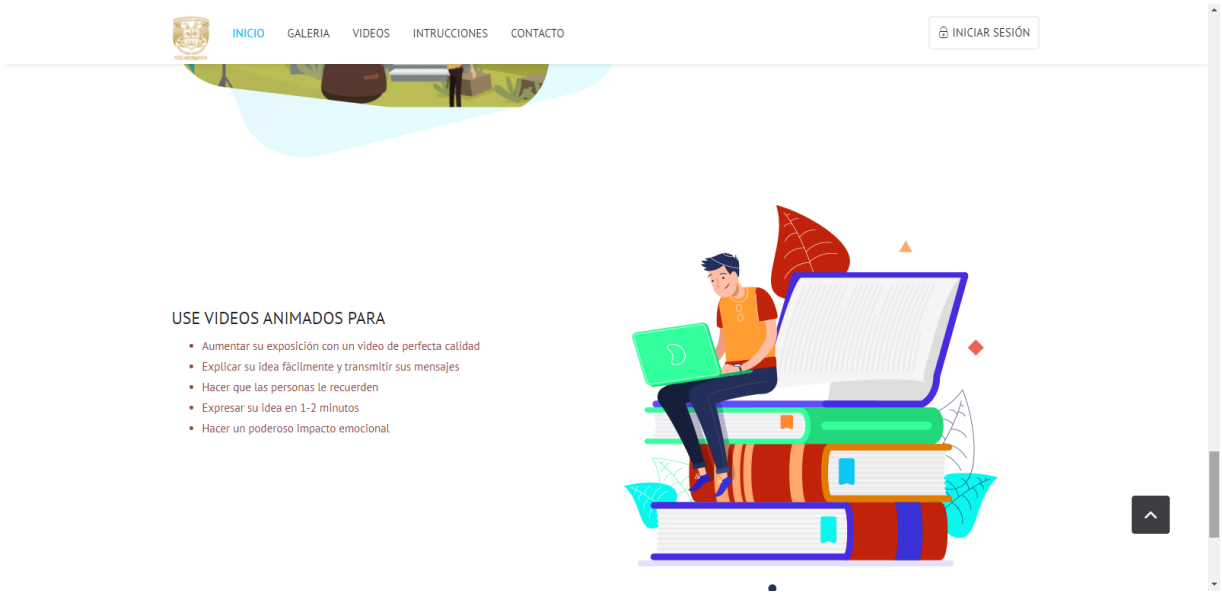


Figura 4.9 Descripción de plataforma.

Y finalmente tenemos un footer o pie de página:

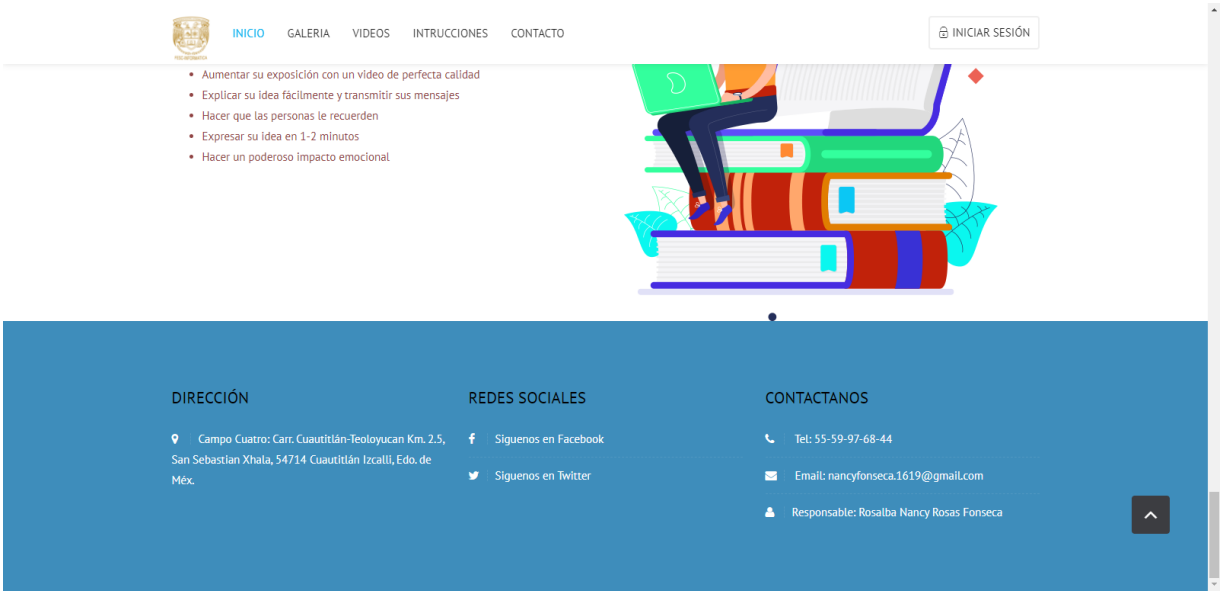


Figura 5.0 Descripción de plataforma.

Ahora, si el usuario inicia un proyecto en la plataforma le desplegara lo siguiente:

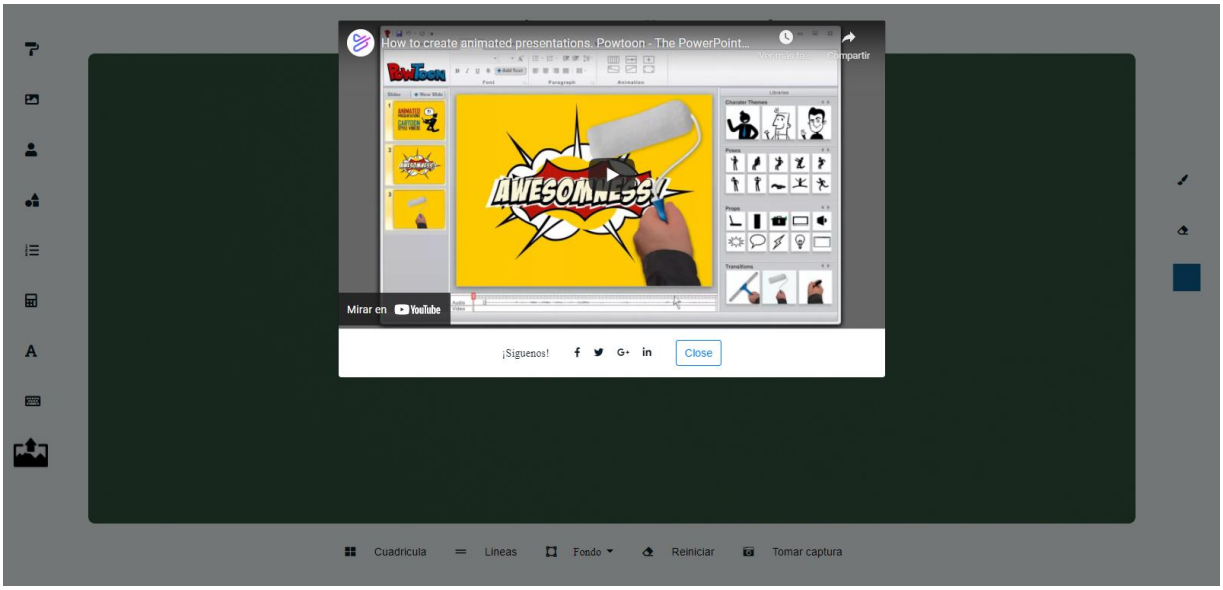


Figura 5.1 Primera vista del lienzo de canvas.

Se tiene como finalidad que la comunidad tenga un preview de cómo utilizar la plataforma.

A la izquierda tendremos una lista grande de recursos para utilizar:

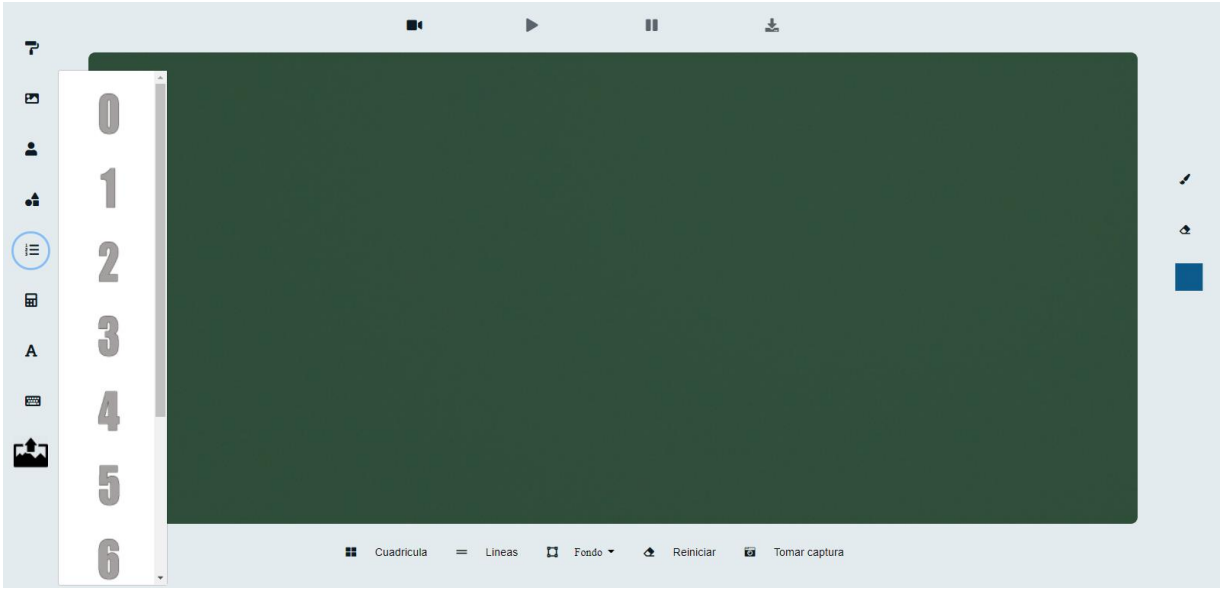


Figura 5.2 Lienzo canvas.

Pueden ser fondos, imágenes, avatares, formas, números, texto, letras, e incluso recursos personales.

Abajo tenemos grid personalizados, así como pizarras y la opción de reiniciar la pizarra u obtener un screenshot.

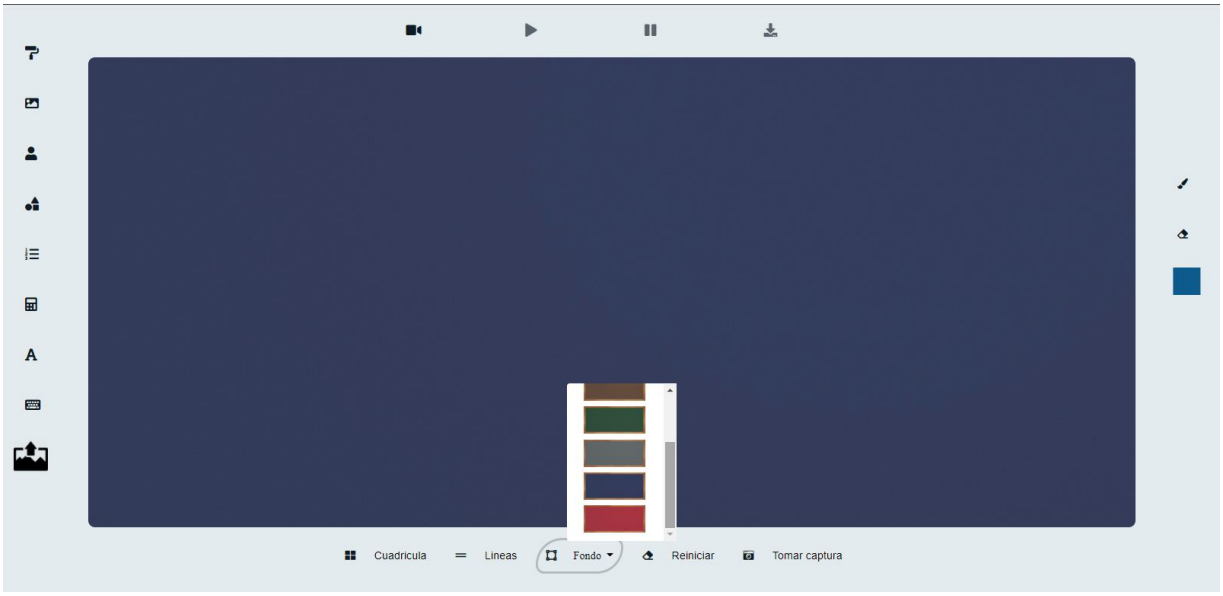


Figura 5.3 Fondos de lienzo.

Un ejemplo rápido de cómo se puede tomar captura es este:

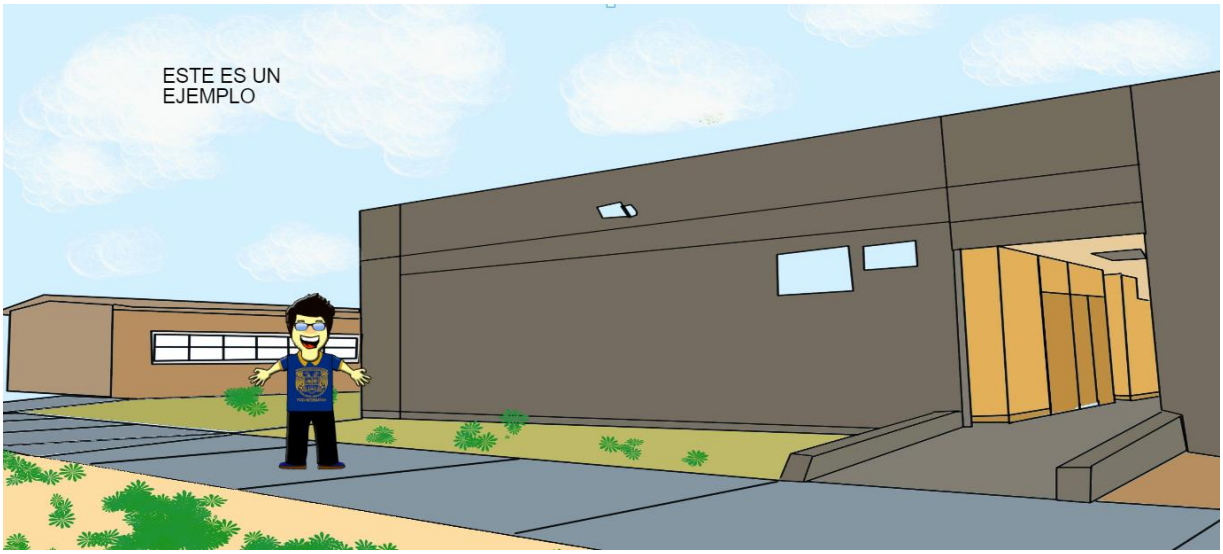


Figura 5.4 Ejemplo de uso de lienzo.

Así mismo en la parte de arriba de puede iniciar una grabación, pausar, detener y descargar para hacer uso del recurso libremente.

Y del lado derecho tenemos un menú libre, para hacer cualquier dibujo en lienzo contando con el recurso de pincel a tamaño personalizable, goma personalizable y paleta RGB de colores:

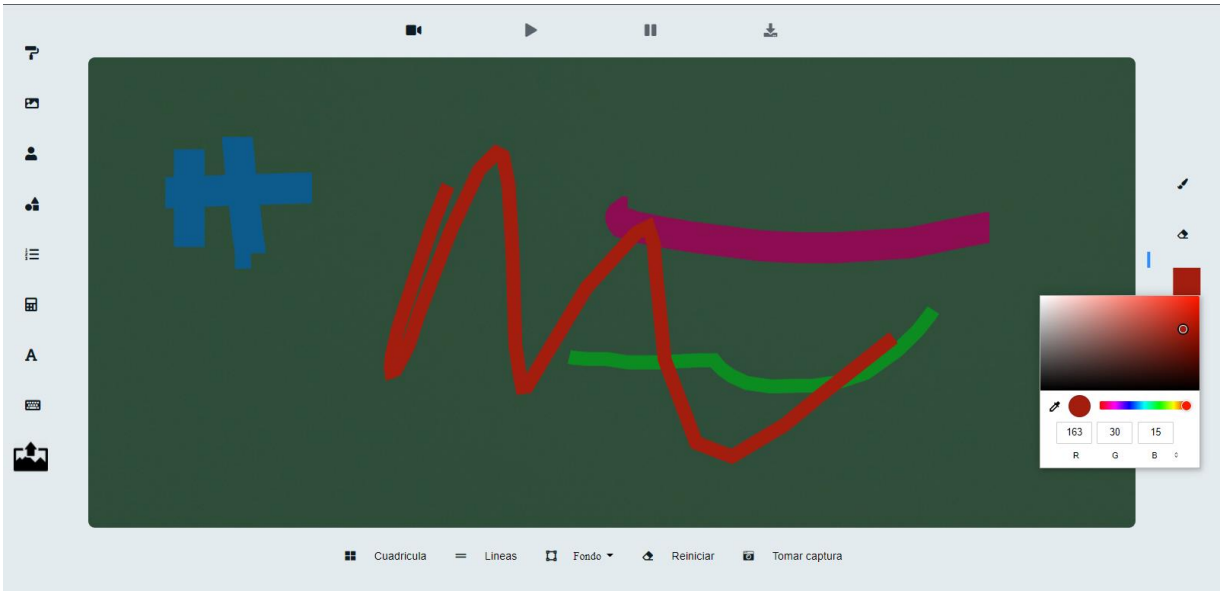


Figura 5.5 Uso de pincel de lienzo.

Aunque sería conveniente detallar el uso de KONVA, que es un framework que nos ayudó mucho a desarrollar esta plataforma, consideró práctico que antes de entrar de lleno a las problemáticas debería mostrar en este espacio un par de ejemplos de código para explicar a groso modo la lógica de trabajo de este framework, por ejemplo, la siguiente función nos ayuda a crear un objeto que pueda pintar de acuerdo a la posición del pincel, para rastrearlo.

```
var isPaint = false;
// var mode = 'brush';
var lastLine;

stage.on("mousedown touchstart", function (e) {
  isPaint = true;
  var pos = stage.getPointerPosition();
  //PRUEBA CAMBIO DE COLOR
  var colorchange, elcolor;
  colorchange = document.querySelector("[type='color']");
  elcolor = colorchange.value;

  //TERMINA PRUEBA CAMBIO DE COLOR
  lastLine = new Konva.Line({
    stroke: elcolor,
    strokeWidth: tamanopin,
    globalCompositeOperation:
      mode === "brush" ? "source-over" : "destination-out",
    points: [pos.x, pos.y],
  });
  layer.add(lastLine);
});

stage.on("mouseup touchend", function () {
  isPaint = false;
});

// and core function - drawing
stage.on("mousemove touchmove", function () {
  if (!isPaint) {
    return;
  }

  const pos = stage.getPointerPosition();
  var newPoints = lastLine.points().concat([pos.x, pos.y]);
  lastLine.points(newPoints);
  layer.batchDraw();
});

mode = "brush";
```

Figura 5.6 Clase de pincel de lienzo.

Y así mismo cuando esta parte de código que nos tomó algo de trabajo adaptar a nuestras necesidades por el desconocimiento del framework, a continuación, explicare las problemáticas en general del proyecto.

```
// URL de las imagenes
var itemURL = "";
document.getElementById("drag-items");
document.addEventListener("dragstart", function (e) {
  itemURL = e.target.src;
});

var con = stage.container();
con.addEventListener("dragover", function (e) {
  e.preventDefault(); // !important
});

con.addEventListener("drop", function (e) {
  e.preventDefault();
  stage.setPointersPositions(e);

  Konva.Image.fromURL(itemURL, function (image) {
    layer.add(image);
    image.position(stage.getPointerPosition());
    image.draggable(true);
    image.name("imagenx");

    image.width(350);
    image.height(250);

    layer.draw();

    image.on("dblclick", function (evt) {
      //evento cuando el usuario de click

      var mensaje;
      var opcion = confirm("¿Deseas eliminar el elemento seleccionado?");
      if (opcion == true) {
        this.remove();
        layer.draw();
      } else {
      }
      document.getElementById("ejemplo").innerHTML = mensaje;
    });
    image.on("contextmenu", function (evt) { }); //fin del contextmenu
  });
```

Figura 5.7 Clase de Konva en uso.

3.3 Problemáticas

Al tener un proyecto en el que no hubo un levantamiento correcto de requerimientos, una planeación y diseño antes del desarrollo, además de la inexperiencia de los desarrolladores podemos desplegar una lista considerable de problemáticas en el mismo, me gustaría empezar por problemáticas en la forma de trabajo, hablemos de como el equipo de desarrollo no trabajo bien con ninguna metodología, al inicio del proyecto contábamos con dos recursos de desarrollo y uno de malajemente, pero conforme el equipo creció los problemas crecieron, en primera instancia intentamos usar scrum, pero debido a la falta de compromiso de los desarrolladores nos encontramos con:

1- Falta de gestión del backlog.
2- Estimaciones sobrepasadas a +300%
3- Estimaciones incorrectas
4- Falta de herramientas virtuales como JIRA.
5- Ausencia del equipo de desarrollo a las Daily Scrum Meeting.
6- Curva de aprendizaje prolongada.

Es por ello que tiramos la idea de trabajar con scrum, entonces intentamos trabajar con XP y aunque también tuvo deficiencias en la forma de trabajo, nos funcionó más por la carga de desarrollo que tiene sobre la de gestión, aunque considero sigue siendo una mala práctica sobreponer el desarrollo sobre la gestión, además nunca hubo un desarrollo de customer cards, lo cual nos puso en desventaja a la hora de plantearnos las funcionalidades o la escalabilidad del proyecto.

Este error únicamente del lado de la metodología ágil, por otro lado, el problema que en mi parecer era y es gravísimo, los desarrolladores se resistieron a utilizar herramientas como GitHub y Git, esto desde luego saca a relucir la inexperiencia y falta de adaptabilidad de los desarrolladores.

Como dato duro, tenemos estas graficas sobre la cantidad de commits y la presencia durante el proyecto:

Figura 5.8 Estadísticas de GitHub.

Aunque la carga de trabajo desproporcionada es visible y por supuesto no está nada alejado de la realidad, es preciso señalar que aunado a la ausencia de los desarrolladores existió la inadaptabilidad a herramientas de desarrollo que no son importantes únicamente sino indispensables, es irracional pensar que un equipo de desarrollo puede trabajar de manera ordenada no usando un repositorio de trabajo o alguna metodología.

Terminando con las problemáticas del lado de la metodología de trabajo y pasando a las problemáticas a la hora de desarrollar el proyecto, específicamente problemáticas técnicas, tengo que destacar que muchos de los errores o malas prácticas que se presentaran a continuación ya fueron refactorizados por mí, sin embargo, la idea es que de los errores que tuve yo y mi equipo la comunidad pueda evitar pasar por ahí.

Empecemos entonces:

Nombres no descriptivos para las clases:

Como podemos visualizar en la imagen de la derecha tenemos clases que no tienen un nombre que por sí mismo pueda describir su funcionalidad, ni siquiera hace una referencia a la usabilidad, o bien no existe un log que indique que hace una clase, y de hecho hay un problema con los índex, ya que tenemos un index.html, un index.php, y un index2.php, lo cual no hace sentido a un orden adecuado de cómo se deben de indexar los menús.

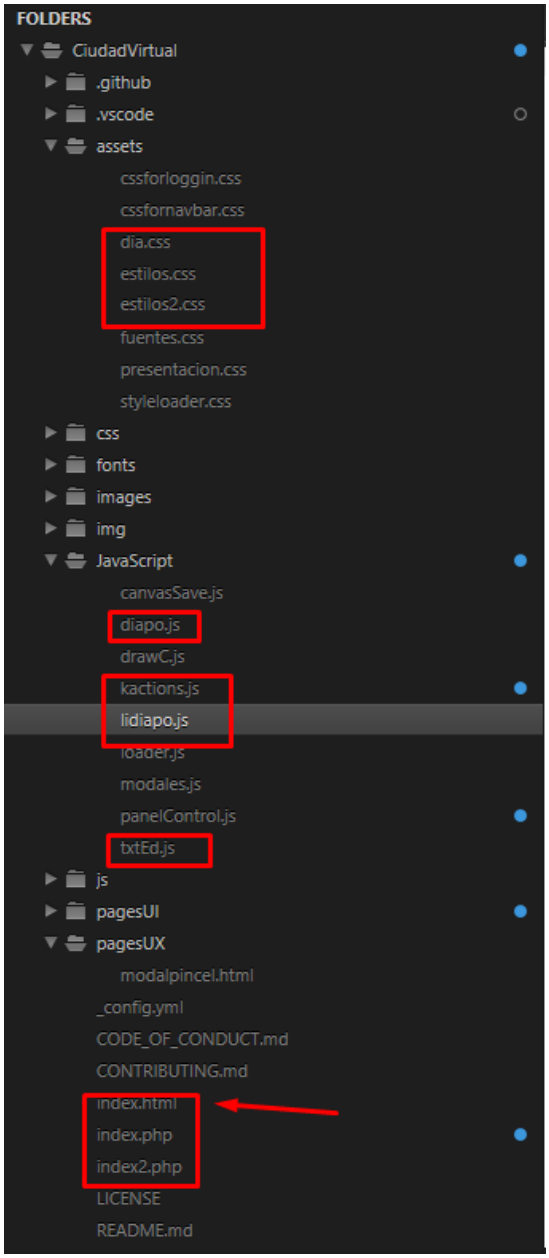


Figura 5.9 Estructura de proyecto.

```
94          background-attachment: fixed;
95
96
97
98
99
100
101
102         border: 3px solid black;
103         width: 86% !important;
104         height: 80% !important;
105         left: 7%;
106
107     }
108
109     #drag-items img {
110         height: 80px;
111     }
112
113
114     div.caja {
115         margin-top: 20px;
116         margin-left: 15%;
117     }
118
119     .pokemon {
120         margin: 10px;
121         height: 80px;
122         width: 80px;
123         max-width: 100%;
124     }
125
```

Figura 6.0 Malas prácticas en clases JS.

En este ejemplo volvemos a señalar el uso de nombres no descriptivos, en el caso de la segunda caja de texto, aunque el código funciona correctamente, es una práctica muy mala hacer este tipo de cosas ya que para darle mantenimiento a este código los desarrolladores tendrían que invertir gran parte del tiempo rastrando el código para entender que hace esa variable con nombre no descriptivo.

Por otro lado, y si bien reitero el código funciona, es necesario recalcar la importancia de hacer una indotación correcta del código, a veces ese tipo de detalles nos puede ahorra tiempo de rastreo o eliminar alguna llave o paréntesis importante.

Código comentado sin eliminar.

```
63
64  //     function draw(e) {
65  //         if (!isDrawing) return; //Stop the function if the user has not pressed left mouse button.
66  //         ctx.strokeStyle = elcolor;  //para cambiar con HSL `hsl(${hue}, 100%, 50%)`;
67  //         ctx.beginPath();
68  //         ctx.moveTo(lastX, lastY);
69  //         ctx.lineTo(e.offsetX, e.offsetY);
70  //         ctx.stroke();
71  //         [lastX, lastY] = [e.offsetX, e.offsetY];
72  //         colorchange.addEventListener("input", actualizar, false);
73  //         //input_color.addEventListener("change", actualizar, false);
74  //         colorchange.select();
75  //         dibujaTrazado.addEventListener("input",cambiar,false);
76  //         dibujaTrazado.select();
77  //     }
78  //     function cambiar(event) {
79  //         if(ctx){
80  //             ctx.lineWidth = document.getElementById("pincelsize").value;
81  //             alert(ctx.lineWidth());
82  //         }
83
84  //      }
85  //     function actualizar(event) {
86  //         // detecta el nuevo color
87  //         elcolor = event.target.value;
88  //         ctx.strokeStyle = elcolor;
89  //         ctx.fillStyle = elcolor;
90  //      }
91
92  //     window.addEventListener('mousedown', (e) => {
93  //         isDrawing = true;
94  //         [lastX, lastY] = [e.offsetX, e.offsetY];
95  //     });
96  //     window.addEventListener('mousemove', draw);
97  //     window.addEventListener('mouseup', () => isDrawing = false);
98  //     window.addEventListener('mouseout', () => isDrawing = false);
99  //     limpiar.addEventListener('click', function(evt) {
100 //         dibujar = false;
101 //         ctx.clearRect(0, 0, cw, ch);
102 //         Trazados.length = 0;
103 //         puntos.length = 0;
104 //       }, false);
105
```

Figura 6.1 Clase de JS comentada.

Cantidad irracional de código comentada, sin eliminar y sin alguna razón justificable, esto lejos de ser una mala práctica nos puede perjudicar a la hora de desplegar el servicio al servidor, nos puede costar espacio que se tiene que pagar, pero no tiene una utilidad para el servicio.

Redundancia de imports:

```
<link rel="stylesheet" href="https://code.jquery.com/ui/1.11.4/themes/smoothness/jquery-ui.css">
<script src="https://code.jquery.com/jquery-1.10.2.js"></script>
<script src="https://code.jquery.com/ui/1.11.4/jquery-ui.js"></script>
```

```
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/
X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="
sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
```

Figura 6.2 Imports de JS en HTML5.

Como podemos ver el descuido de no verificar que estamos importando si nos puede generar un conflicto de versiones.

Y seguido de ello:

La falta de debug, fue algo que definitivamente predomino en el desarrollo de la plataforma.

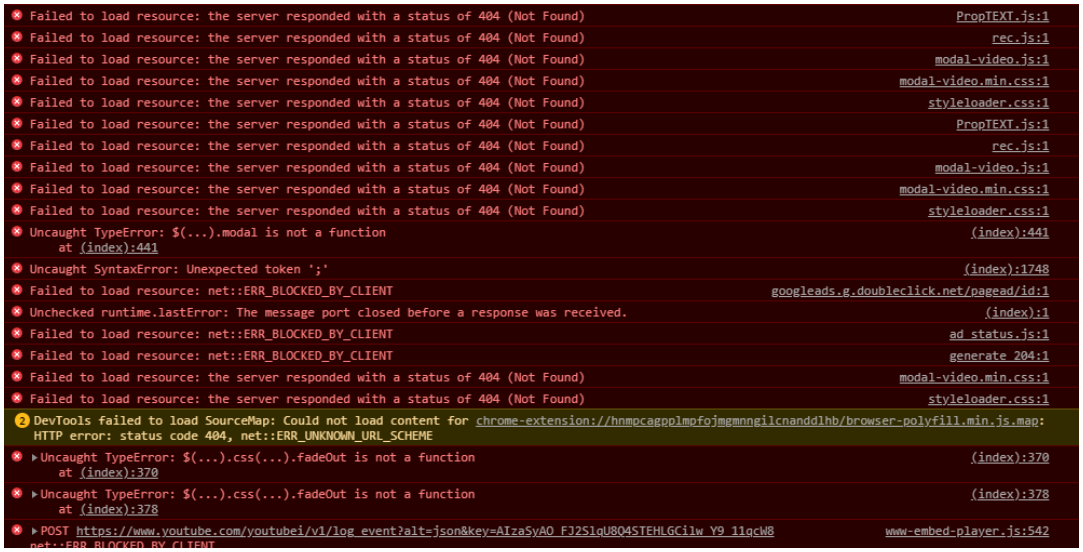
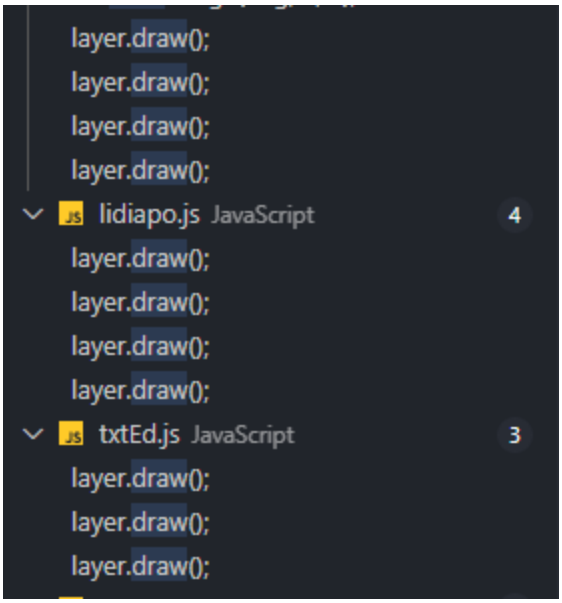


Figura 6.3 Debug de JS.

No existió un monitoreo constante de la aplicación en cuanto a bugs se refiere, es decir, si, cumple su funcionalidad de manera “correcta” sin embargo no de la manera más optima posible, y mientras exista una manera más optima los desarrolladores deben de buscar llegar al punto más alto de optimización.

Código espagueti y ausencia de POO.



Como podemos ver en este caso, se utiliza un método en diferentes clases, el problema aquí es que ese método esta declarado en cada clase, es decir no existe una clase padre que herede a esas clases esa funcionalidad, esto ya lejos de una mala práctica, se convierte en un problema serio a la hora de refactorizar, ya que es un tiempo destinado a resolver algo que se pudo haber hecho bien desde el inicio.

Figura 6.4 Redundancia de métodos.

No comentarios de ningún tipo.

En esta clase que tiene funciones medianamente complejas debería de existir una documentación mínima, pero no la hay a excepción de un par de comentarios que colocamos algunos.

```
    diw = "container"+ constan;
    constan= constan+1;
    respuesta.appendChild(newDiv);
    console.log(diw);
var width = window.innerWidth;
var height = window.innerHeight;
var widthCanvas = 57;
var heightCanvas = 75;
var stage = new Konva.Stage({
  container: diw,
  width: (width / 102) * widthCanvas,
  height: (height / 100) * heightCanvas-20,
});
var layer = new Konva.Layer();
layer.id('canvas');
stage.add(layer);


stage.getContainer().style.border = '10px solid black';
stage.getContainer().style.background = '#D4D4D4';
stage.getContainer().style.width = widthCanvas + "%";
stage.getContainer().style.height = heightCanvas + "%";
// URL de las imagenes
var itemURL = '';
document.getElementById('drag-items')
document.addEventListener('dragstart', function (e) {
  itemURL = e.target.src;
});

var con = stage.container();
con.addEventListener('dragover', function (e) {
  e.preventDefault(); // !important
});

con.addEventListener('drop', function (e) {
  e.preventDefault();
  // now we need to find pointer position
  // we can't use stage.getPointerPosition() here, because that event
  // is not registered by Konva.Stage
  // we can register it manually:
  stage.setPointersPositions(e);
```

Figura 6.5 Mala estructura de comentarios.

No refactorización por parte del equipo:

Lamentablemente en la comunidad del desarrollo de software existe una frase que dice: “Si funciona, ni lo toques”, y que como ya vimos no siempre que funcione es sinónimo de trabajo bien hecho, y lamentablemente gracias a esa frase existió una negativa de los desarrolladores por refactorizar su propio código, lo cual definitivamente es una mala práctica.

Falta de testing:

Para concluir con este capítulo considero que es preciso señalar que el testing es una parte fundamental que como desarrolladores no le tomamos importancia, sin embargo, con este proyecto como experiencia, y con lo anteriormente expuesto, recomendaría ampliamente hacer de mínimo pruebas no automatizadas a tu sistema, de hecho con pruebas regresivas en ambiente dev y ambiente de producción es más que suficiente para la mayoría se sistemas, y lo ideal sería no desechar esta idea y de hecho tomar con firmeza el desarrollo iterativo:

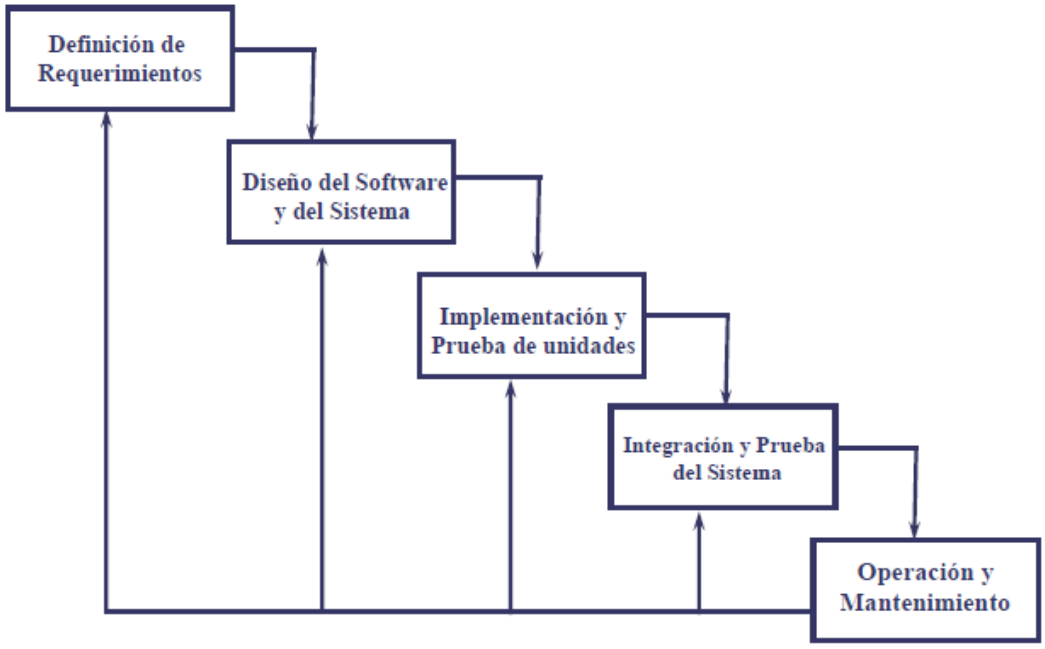


Figura 6.7 Iteración del desarrollo de software.

3.4 Recomendaciones

Después de un intenso capitulo, donde señale muchas de las deficiencias del desarrollo del proyecto, viene la parte en donde la investigación en conjunto con mi experiencia desarrollando software, y lo aprendido con este proyecto emitiré algunas recomendaciones para no solo este proyecto sino para cualquier proyecto de desarrollo de software, esto con la finalidad de que la comunidad estudiantil no se llene de malas prácticas y posteriormente las lleven al mundo laboral y que no tengan que pasar por los malos ratos que pase yo.

Empecemos por las recomendaciones del lado de la gestión de un proyecto:

1. Es importante conocer a tu equipo de desarrollo sin importar en la etapa en la que estés, conocer las fortalezas y debilidades de tus compañeros te ayudara a tener una mejor estimación de las tareas a realizar.
2. Solicita los recursos necesarios para desempeñar las tareas.
3. “Zapatero a tu zapato” muchas veces como desarrolladores tenemos el gusto por realizar muchas tareas o nos sentimos con la capacidad de desarrollar las tareas de los demás, lo cierto es que esto evitara que realices tus tareas de la mejor manera posible y hará que la carga de trabajo sea inequitativa.
4. Utiliza una metodología ágil, aunque no es regla, es cierto que actualmente son muy eficientes para llevar a cabo un proyecto de desarrollo de software.
5. Comprométete, e incentiva a tus compañeros a comprometerse con el desarrollo, en especial con las etapas de gestión como pueden ser la gestión de backlog o las daily, muchos desarrolladores las ven como aburridas y pérdida de tiempo, sin embargo, son muy necesarias para la correcta gestión del proyecto.
6. Aunque no sea agradable tenemos que aceptar que existen compañeros que simplemente no tienen interés en desempeñar sus actividades, si es el caso y existe alguna dependencia de tus actividades con las de él, coméntale la situación a tu PM o en todo caso a la autoridad competente.

En cuanto a las recomendaciones técnicas:

1. Si es parte de tus actividades realiza un correcto levantamiento de requerimientos, no importa el tiempo que te lleve, una técnica para levantar requerimientos es simular la usabilidad del sistema, ve físicamente al lugar donde el software trabajara, interactúa con quien el software tenga que interactuar, esto te permitirá descubrir muchos requerimientos que no te dijeron explícitamente.
2. Realiza un diseño conceptual correctamente, que no queden monedas al aire, todo es importante, si es parte de tus actividades documéntate correctamente para realizar una arquitectura correcta.
3. Al momento de empezar con la aplicación técnica, realiza una lluvia de ideas para que elijan las tecnologías correctas, para ello te recomiendo mantener una posición neutral por muy difícil que esto resulte, sabemos que como desarrolladores tenemos tecnologías favoritas, sin embargo, es preciso señalar que como profesionistas debemos de ofrecer soluciones al usuario basados en sus necesidades y los recursos.
4. Evalúen la posibilidad de usar un framework, odiados por muchos y amados por otros, recuerda que no existen malas o buenas tecnologías, solo desarrolladores buenos o malos, por poner un ejemplo, después de tiempo y reconsiderando las especificaciones nos dimos cuenta que era viable usar un framework web como flask o laravel que nos hubiera ayudado a utilizar MVC, para crear código legible y más ordenado.
5. Aprendan y aplique modelos y patrones de diseño , además de tener buenas prácticas de desarrollo como documentar su código, nombres descriptivos, código auto documentable, etc, no todo es aprender lenguajes de programación.
6. Realizar un debug constante en tu desarrollo.
7. Realiza testing constantemente a tu desarrollo.

8. Utiliza un IDE o Editor de texto y los plugins necesarios para hacer tareas automatizadas y evita perder tiempo en cosas como indentar tu código.
9. Revisa la documentación de la tecnología que estés aplicando o si llegas a tener un problema primero revisa la documentación antes que StackOverflow.
10. Asigna roles a equipos JR, identifica sus hardskills y envía a los recursos al área correspondiente, si tiene habilidades de UI colócalo en frontend, si tiene habilidades en el server y modelo envíalo a backend, y si su fuerte son los datos colócalo cono DBA.
11. Cuando realices tu propuesta de solución considera que esta sea viable técnicamente, a veces comentemos el error de prometer cosas que no podemos cumplir.
12. Refactoriza siempre que puedas tu código, refactorizar es encontrar cada vez más formas de perfeccionar lo que creamos.
13. Elimina dichos y malas prácticas, no solo se trata de “Tirar código”, se trata de hacerlo bien.

Capítulo IV:

Conclusiones

Finalmente generare conclusiones en base a dos aspectos, uno de ellos será el marco de trabajo o metodología empleada (XP) y el otro será la implementación técnica en el proyecto (Stack tecnológico usado).

Usualmente XP es una metodología poco usada actualmente debido a el enfoque técnico que tiene, se encuentra en desventaja con metodologías agiles modernas como Scrum o Scrumban que también buscan llevar una comunicación estrecha con el cliente para iterar la implementación del proyecto en sprints los cuales deberían durar 4 semanas en estos hay previamente una “ceremonia” llamada Sprint Planning, en esta ceremonia se revisa el product backlog y el equipo selecciona los producto backlog ítems que desarrollaran en el sprint, durante esta reunión también el producto owner tendrá que presentar el product backlog actualizado, cabe destacar que durante este periodo no es necesaria la presencia de los stakeholders, porque esto es un proceso meramente en el equipo de desarrollo y queda definida la sprint goal.

Otra parte de scrum muy importante es el scrum daily, la cual es o debería ser empleada día a día en un periodo corto, usualmente 15 minutos (esto depende mucho de la cantidad de integrantes), y la finalidad es dar una breve explicación sobre que ha hecho, que hará hoy y si tiene algún problema o dependencia para lograrlo.

Cada viernes usualmente el equipo suele realizar un sprint review en el que finalmente se presentan los avances a los stakeholders, aquí se inspeccionan los avances y se define el rumbo de los cambios que puedan llegar a existir, ya que, aunque el producto owner está presente, estos avances son presentados por el equipo de desarrollo.

Al final de cada sprint hay un sprint retrospective la cual se realiza en conjunto con todo el equipo, para analizar en conjuntos que se hizo bien, que se hizo mal, y como podemos mejorar para el siguiente sprint, usualmente este espacio es más didáctico ya que la finalidad es recolectar ideas, usualmente se usa un espacio con post-its.
Es importante entender cómo funciona scrum, de esta forma podremos entender que muchos aspectos de scrum no hubieran podido ayudar a llevar de mejor manera el proyecto, por ejemplo, tener un sprint goal, hubiera sido de vital importancia para que se cumplieran objetivos claros en poco tiempo, además llevar scrum dailys nos hubiera servido para día a día revisar el avance de un proyecto y en caso de que tuvieran stoppers se hubieran podido eliminar en poco tiempo, evitando que esto retrase la entrega del proyecto como paso en muchas ocasiones, por ejemplo otra cosa que podríamos haber implementado es el uso de un tablero Kanban , lo cual nos hubiera llevado a tomar una metodología tipo scrumban, para tener la visibilidad de que tareas están hechas, cuales están detenidas, cuales están en pruebas y cuales están en QA.

Otra mejoría que hubiera sido de vital importancia es la iteración de trabajo es la falta de deadlines ya que constantemente el trabajo que no se terminaba a tiempo se empujaba constantemente, lo cual hacía que esto se convirtiera en una bola de nieve que cada vez era más complicada de solucionar, al mismo tiempo la falta de personas que se dedicaran a realizar pruebas a las nuevas funcionalidades, es por ello que eventualmente teníamos problemas porque el mismo equipo de desarrollo tenía que dedicar tiempo para poder realizar las pruebas de su mismo desarrollo, por ello había fallas en las integraciones , ya que el equipo de desarrollo no realizar pruebas que rompieran intencionalmente su desarrollo .

En cuanto a viabilidad técnica, al ser un proyecto por alumnos que no tenían experiencia en la industrial del software decidimos usar un stack de tecnología base como LAMP + HTML5,CSS3,JS,PHP,KONVA JS , WEBRTC siendo una aplicación monolítica, además de tener en muchas clases código spaguetti, lo cual hizo cada vez más difícil que implementáramos nuevas características, que resolviéramos bugs, o la

refactorización del mismo, posteriormente pudimos llegar a la conclusión de que lo ideal hubiera sido implementar un framework que utilizara tal vez el MVC con la finalidad de no escribir código spaguetti, como NODE JS, Laravel etc, y nos dimos cuenta que ciertas funcionalidades se podían dividir en microservicios para no tener todo centralizado y además para evitar que la carga del sitio sea larga, también para que el equipo pudiera trabajar en cosas distintas al mismo tiempo, también concluimos como equipo que otra mejora hubiera sido una implementación cloud, actualmente los 3 grandes proveedores de nube (Google, Amazon, Microsoft) tienen soluciones que nos pueden ayudar a :

- Implementar los servicios de manera continua y segura
- Evitamos que tengamos el problema de que cuando un servicio tiene más demanda se empiezan a saturar, por el contrario, en una nube podría tener auto escalado.
- Pago por consumo, es decir cuando el servicio no ocupe muchos recursos el pago será menor.

Teniendo en cuenta esto, y que Google regala 300 dólares para pruebas gratuitas, entonces podríamos usar un servicio autogestionado como Google App Engine y únicamente nos tendríamos que preocupar por desplegar los servicios ya que App engine tiene la capacidad de aumentar los recursos cuando las consultas o visitas son más.

Por otra parte, podríamos escalar a una serie de servicios autogestionados los cuales se alojen a la nube y en los cuales trabaje el equipo de desarrollo para únicamente agregar nuevas características, porque al final nosotros tenemos únicamente la finalidad de ofrecer software como servicio, además de usar una metodología más iterativa como scrum la cual nos pueda llevar a un marco de trabajo ágil.

Finalmente, dadas las problemáticas técnicas y metodológicas del proyecto, podemos respaldar la hipótesis sobre la deficiencia que tiene el alumno de informática a la hora de emplear sus conocimientos teóricos.

Bibliografía

Pressman, R. (1997). Ingeniería del software: un enfoque práctico. México: McGraw-Hill.

Fernández, V. (2006). Desarrollo de sistemas de información: una metodología basada en el modelado. México: UPC.

Kendall, K. E. (2005). Análisis y diseño de sistemas (6.ª ed.). México: Pearson

Bedini,A. (2006). Gestión de Proyectos de Software. Febrero 17,2020, de Universidad Técnica Federico Santa María Sitio web: https://www.inf.utfsm.cl/~guerra/publicaciones/Gestion%20de%20Proyectos%20de%20Software.pdf

IBM. Guía de iniciación a la ingeniería de software y sistemas. Febrero 17,2020, de IBM Sitio web: https://www.ibm.com/support/knowledgecenter/es/SSQQC5_6.0.0/com.ibm.rational.sse.doc/topics/c_getting_started.html

Bleakley G, Collyer K, & Scouler J. (2013). Buenas prácticas para el desarrollo de sistemas y software. Febrero 17,2020, de IBM Sitio web: https://www.ibm.com/developerworks/ssa/rational/library/systems-software-lifecycle-development/index.html

Davis, A. (1995). 201 Principles of software development. University of Colorado: McGraw-Hill.

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. Febrero 17,2020, de IEEE Sitio web: https://pdfs.semanticscholar.org/dce9/9209120ebed7f5d68e3644fdcd160d4c366c.pdf?_ga=2.105784320.228216168.1581960208-927189479.1581960208

Edelsys Hernández Meléndrez. (2006). Como escribir una tesis. Febrero 17,2020, de Pontificia Universidad Católica de Valparaíso Sitio web: http://biblioteca.ucv.cl/site/servicios/documentos/como_escribir_tesis.pdf

Méndez. (2018). Principales herramientas CASE del mercado y su uso. Febrero 17,2020, de Universidad de Murcia Sitio web: https://www.um.es/docencia/barzana/IAGP/Enlaces/CASE_principales.html

Dirección General de Administración Electrónica y Tecnologías de la Información. Metodología de Ingeniería del Software para el desarrollo y mantenimiento de sistemas de información del Gobierno de Extremadura. Febrero 17,2020, de Gobierno de Extremadura Sitio web: http://www.juntaex.es/filescms/con01/uploaded_files/dgaeti/IngenieriaSoftwareGobex.pdf

James, A. (1994). Análisis y Diseño de Sistemas. México: Mc-Graw Hill.

Universidad de Pamplona. Análisis y Diseño de Sistemas de Información. Febrero 17,2020, de Universidad de Pamplona Sitio web: http://www.unipamplona.edu.co/unipamplona/portalIG/home_109/recursos/octubre2014/administraciondeempresas/semestre7/11092015/analisisydisenosistinformacion.pdf

Grupo ISSI. (2003). Metodologías Ágiles en el Desarrollo de Software. Febrero 17,2020, de UNAM Sitio web: http://fcaenlinea1.unam.mx/anexos/1728/Unidad_1/u1_act2_1.pdf

Gómez. (2001). Análisis de requerimientos. UAM: Publidisa Mexicana S. A. de C.V.

Menéndez, R. & Barzanallana, A. (febrero 29, 2020). Principales herramientas CASE del mercado y su uso. Febrero 01, 2022, de Universidad de Murcia Sitio web: https://www.um.es/docencia/barzana/IAGP/Enlaces/CASE_principales.html

Menéndez, R. &, Barzanallana, A. (febrero 29, 2020). Ingeniería del software. Febrero 01, 2022, de Universidad de Murcia Sitio web: https://www.um.es/docencia/barzana/IAGP/Iagp1.html

Fuentes, J. (2003). Realidad virtual aplicada al tratamiento del trastorno de lateralidad y ubicación espacial. Febrero 01, 2022 (tesis de Licenciatura, Universidad de las Américas, Puebla)- Recuperado de http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/fuentes_k_jf/

Servicio de Desarrollo de Proyectos. (2014). Metodología de Ingeniería del Software para el desarrollo y mantenimiento de sistemas de información del Gobierno de Extremadura. Febrero 01, 2022, de Dirección General de Administración Electrónica y Tecnologías de la Información Sitio web: http://www.juntaex.es/filescms/con01/uploaded_files/dgaeti/IngenieriaSoftwareGobex.pdf

González, P. & Avalos, M. (2012). Lista de herramientas CASE. Febrero 01, 2022, de UDELP, Ingeniería Sitio web: https://sites.google.com/site/herramientascaseudelp/assignments

Coordinación de Universidad Abierta y Educación a Distancia de la UNAM. (2017). Desarrollo de sistemas. Febrero 01, 2022, de UNAM, SUAyED Sitio web: https://programas.cuaed.unam.mx/repositorio/moodle/pluginfile.php/1150/mod_resource/content/1/contenido/index.html

Pressman, R. (2010). Ingeniería del Software un enfoque práctico. México: Mc Graw Hill. Recuperado de: http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF

Bedini, A. & Guerra, L. (2005). Gestión de Proyectos de software. Valparaíso, Chile: Universidad Técnica Federico Santa María. Recuperado de : https://www.inf.utfsm.cl/~guerra/publicaciones/Gestion%20de%20Proyectos%20de%20Software.pdf

Aguilar, R. & Díaz, J. (2015). La Ingeniería de Software en México: hacia la consolidación del primer programa de licenciatura. CONAIC, II, pp. 6-12. Recuperado de https://conaic.net/revista/publicaciones/2doVol2015Articulo%201.pdf

Zarazaga, F. & Alfonso M. (2003). La Ingeniería del Software en el currículo del Ingeniero en Informática. Febrero 01, 2022, de Novática: Revista de la Asociación de Técnicos de Informática Sitio web: http://www.dccia.ua.es/~eli/novatica03.pdf

Vargas, B-. (2007). Panorama general de las "Herramientas Case" (Tesis de Licenciatura, Universidad Autónoma del Estado de Hidalgo) Recuperado de: https://repository.uaeh.edu.mx/bitstream/bitstream/handle/123456789/11134/Panorama%20general%20de%20las%20herramientas%20CASE.pdf?sequence=1

Cruz, I. & Fernández C. (2003). UMLGEC++: Una Herramienta CASE para la Generación de Código a partir de Diagramas de clase UML. Febrero 01, 2022, de Universidad Tecnológica de la Mixteca Sitio web: https://www.utm.mx/~caff/doc/CASE_UML_ANIEI2003.pdf

Cascio, B. (2019). Requerimientos, Técnicas de Especificación y metodologías Ágiles. Febrero 01, de GitHubGist Sitio web: https://gist.github.com/brunocascio/09ba6c90842948bfb9ad

IEEE. (1998). Especificaciones de los requisitos del Software. Febrero 01, 2022, de IEEE Sitio web: https://www.ctr.unican.es/asignaturas/is1/IEEE830_esp.pdf

Parker, K. (2015). Mejores prácticas de ingeniería de requisitos. Febrero 01, 2022, de MicroFocus Sitio web: https://www.microfocus.com/es-es/media/white-paper/requirements_engineering_best_practices_wp._es.pdf

Corvo, H. (2020). Modelo espiral: historia, características, etapas, ejemplo. Febrero 01, 2022, de Lifeder Sitio web: https://www.lifeder.com/modelo-espiral/

Agilemanifiesto. (2001). Manifesto for Agile Software Development. Febrero 01, 2022, de Agilemanifiesto.org Sitio web: https://agilemanifesto.org/

Cuomo V. & Castares M. (2016). Calidad de Software. Febrero 01, 2022, de Universidad Nacional del Sur Sitio web: https://cs.uns.edu.ar/~virginia.cuomo/calidad-2016/downloads/CalidadSW-2016-Teoria05-Lean.pdf

Drummond C. (2018). Scrum: Aprende a utilizar scrum con lo mejor de é. febrero 01, 2022, de ATLASSIAN Sitio web: https://www.atlassian.com/es/agile/scrum

Kanbanize. (S.F). Qué es Kanban: Definición, Características y Ventajas. Febrero 01, 2022, de Kanbanize Sitio web: https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban

GoogleCloud. (2007). App Engine. Febrero 01, 2022, de Google Cloud Sitio web: https://cloud.google.com/appengine

Google Cloud. (2007). Cloud Storage. Febrero 01, 2022, de Google Cloud Sitio web: https://cloud.google.com/storage

Sliger, M. (2011). Agile project management with Scrum. Febrero 01, 2022 de PA: Project Management Institute Sitio web: https://www.pmi.org/learning/library/agile-project-management-scrum-6269

Maynard, C. (S.F). Aprender a usar kanban con Jira Software. Febrero 01, 2022, de ATLASSIAN Sitio web: https://www.atlassian.com/es/agile/tutorials/how-to-do-kanban-with-jira-software

Rehkopf, M. (S.F). Historias de usuario con ejemplos y plantilla. Febrero 01, 2022, de ATLASSIAN Sitio web: https://www.atlassian.com/es/agile/project-management/user-stories#:~:text=usuario%20del%20software.-

,Una%20historia%20de%20usuario%20es%20una%20explicaci%C3%B3n%20general%20e%20informal,un%20valor%20particular%20al%20cliente

Rehkopf, M. (S.F). Epics ágiles: definición, ejemplos y plantillas. Febrero 01, 2022, de ATLASSIAN Sitio web: https://www.atlassian.com/es/agile/project-management/epics#:~:text=Un%20epic%20es%20una%20gran,de%20ellos%20en%20varios%20tableros

Lefort, A. (2019). Diferencias entre Cloud vs On Premise. Febrero 01, 2022, de Teamnet Sitio web: https://www.teamnet.com.mx/blog/cloud-vs-on-premise#:~:text=Una%20de%20las%20mayores%20ventajas,casan%20a%20un%20solo%20equipo

Azure. (S.F). ¿Qué es SaaS? Software como servicio. Febrero 01, 2022, de Azure Sitio web: https://azure.microsoft.com/es-mx/overview/what-is-saas/

Gómez M. (2011). Notas del curso: Análisis de requerimientos. México: Universidad Autónoma Metropolitana.

Fernández V. (2006). Desarrollo de sistemas de información. Barcelona: Editions de la Universitat Politécnica de Catalunya SL

Escuela de Informática. (2015). Análisis y Diseño de Sistemas “Ingeniería y gestión de requerimientos” febrero 01, 2022 de: Universidad Tecnológica de Chile. Sitio web.

Gold, J-. (2019). Ingeniería de software aplicada a la elaboración de material multimedia para plataformas de e-learning (Tesis de Ingeniería, Universidad Nacional Autónoma de México)

IEEE. (1990). IEEE Standard Glossary of software Engineering Terminology. New York: IEEE.

Arguello, Gutiérrez, A, & Rodríguez, J-. (2013). Stakeholders en desarrollo web (Tesis de Ingeniería, Universidad Argentina de la Empresa)

Letelier, P & Sánchez E. (2003). Metodologías ágiles en el desarrollo de software. Alicante, España: Grupo ISSI.

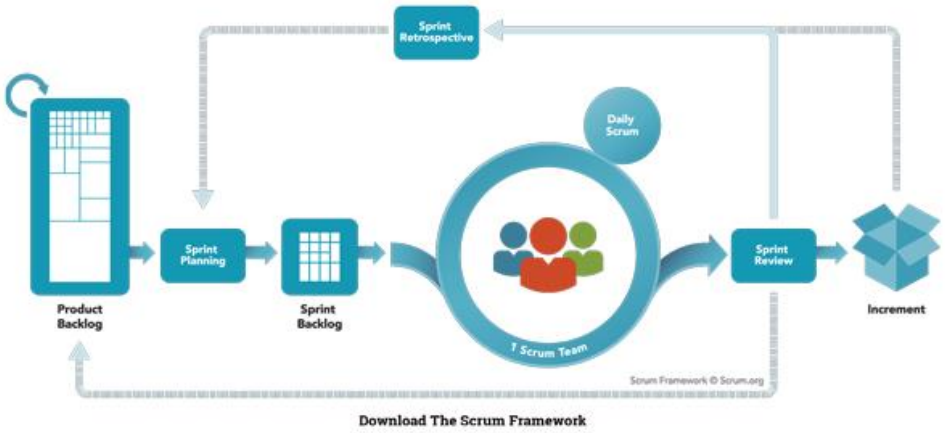
González, Castro, V. (2000). Guía de autoevaluación de la categoría de procesos de ingeniería de software bajo el estándar ISO-15504 (SPICE) (Tesis de ingeniería, Facultad de Ingeniería, Universidad Nacional Autónoma de México)

Universidad de Pamplona. (2014). Análisis y diseño de sistemas de información (Plan de estudios, Universidad de Pamplona)

Anexos

Anexo A.

Estructura de SCRUM.



Download The Scrum Framework

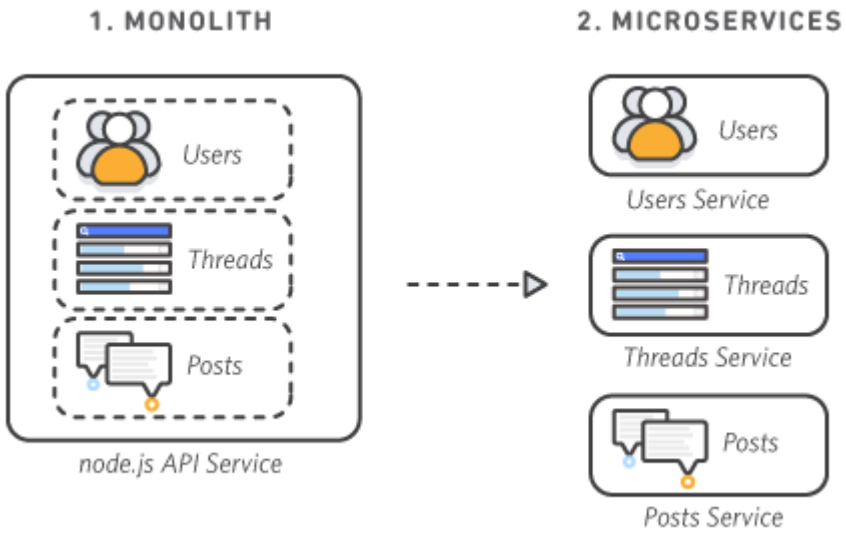
Anexo B.

Historia de usuario.

Historia de Usuario	
Número: 1	**Usuario:** Cliente
Nombre historia: Cambiar dirección de envío	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Puntos estimados: 2	**Iteración asignada:** 1
Programador responsable: José Pérez	
Descripción: Quiero cambiar la dirección de envío de un pedido.	
Validación: El cliente puede cambiar la dirección de entrega de cualquiera de los pedidos que tiene pendientes de envío.	

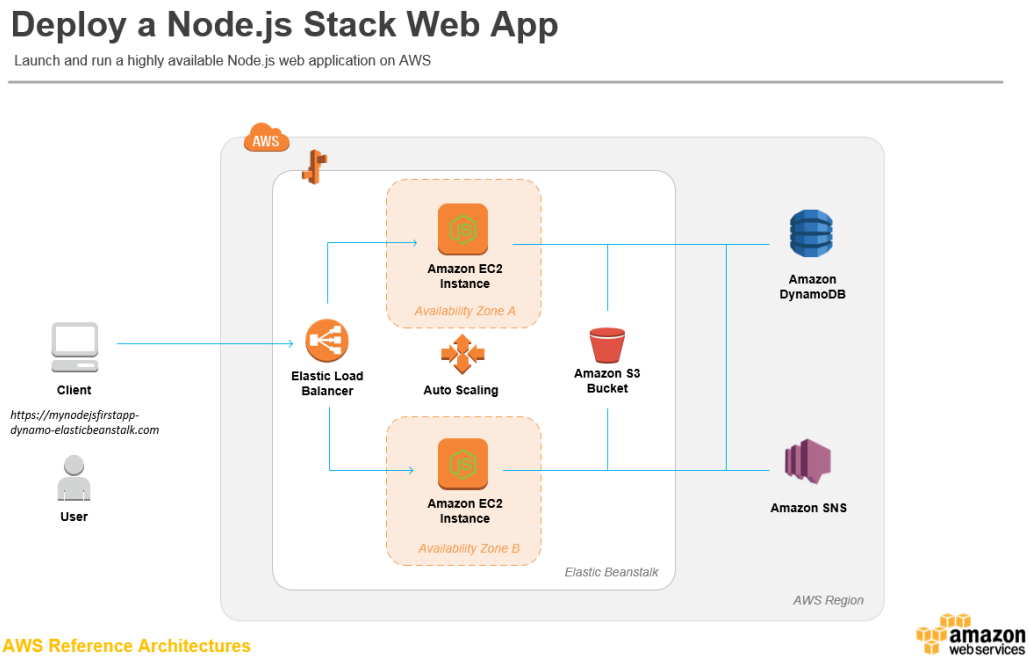
Anexo C.

Diferencias de monolítica a microservicios.



Anexo D.

Ejemplo de web fullstack montada en AWS.



Anexo E.

Kanban en desarrollo de software.