PLAN DE PROYECTO

BestTaste

Por: Teresa De La Vega, Alejandro Garrido, Jorge Martín, Roi López, Francisco Adrián Fonta, Álvaro Tejedor, Jhon Fredy Jaramillo y Samuel Martínez

1. INTRODUCCIÓN

En este documento se expone el plan de proyecto de BestTaste, permitiendo que se vea cómo se va a llevar a cabo de manera clara y concisa. Se desarrollarán sus objetivos, sus riesgos, ámbitos, estimaciones y su estructura de trabajo.

1.1 Propósito del plan

El objetivo de este proyecto es diseñar una aplicación que permita a un usuario recibir y dar recomendaciones de música según el género, uniéndole con una comunidad de gente que comparte sus mismos gustos musicales. Para una buena planificación, lo haremos aplicando las técnicas de ingeniería del software a nuestro proyecto para así llevar a cabo buenas técnicas de construcción de software y evitar desorganización, cumplir la fecha límite y lograr un resultado satisfactorio. Se especificará a continuación el ámbito del proyecto y los objetivos del mismo, así como el modelo de proceso a seguir a lo largo de la elaboración.

1.2 Ámbito del proyecto

A continuación, se define el ámbito del proyecto en función del contexto, que va dirigido a la organización del mismo, considerando varios factores en el entorno en el que se desarrollará el proyecto.

De cara al desarrollo del documento SRS se usará la Especificación de Requisitos según el estándar de IEEE 830 (std: SRS_IEEE_830).

1.2.1 Declaración del ámbito

Este punto queda definido en la SRS, sin embargo, para facilitar la lectura al lector, incluiremos un resumen:

BestTaste será una red social que busque ayudar a sus usuarios a encontrar música acorde con sus gustos, a partir de recomendaciones hechas por el mismo u otros usuarios, las cuales estarán limitadas a una cierta cantidad por día, que incluirán una previsualización o *preview* de 15 segundos de la canción. El usuario podrá interactuar con dichas recomendaciones mediante *likes* y *comments,* que, además, ayudarán al sistema con las recomendaciones que hace a cada usuario para, así, ajustarlas a sus gustos.

Contaremos, además, con sponsors que publiquen anuncios en la aplicación, y estos anuncios aparecerán cada 5 recomendaciones para el usuario casual, pero proporcionaremos la posibilidad de suscribirse a BestTaste+ por el precio de 9,99 al mes para quitar los anuncios y, además, poder hacer más recomendaciones en un mismo día, además de otras ventajas.

El propósito de BestTaste es ayudar a usuarios a compartir su cultura musical para que así otros puedan usarla para ampliar la suya propia.

1.2.2 Funciones principales

Las funciones del proyecto se resumen en: “Alta” o “Añadir”, que son las encargadas de crear/registrar un elemento del módulo correspondiente, “Modificar” que permite hacer cambios en nuestro caso sobre la propia cuenta del usuario o la lista de canciones, “Baja” o

“Borrar”, que consiste en borrar la información de la base de datos y finalmente “Mostrar” que como bien indica su nombre, será la función encargada de mostrar lo que corresponda. Estas funciones las desarrollamos más a fondo en la sección 2.1 Funciones del producto de la SRS

Usuario:

- **Dar de alta**: Registrar al usuario en la base de datos.
- **Dar de baja**: eliminar la cuenta.
- **Log in**: Acceso a cuenta ya existente en la base de datos.
- **Log out**: Salir de la cuenta.
- **Modificar**: cambiar los datos que el usuario desee.
- **Mostrar**: para mostrar la lista de usuarios.
- **Añadir canción**: Añade una canción, si no existía ya, a la lista de canciones
- **Darse de alta en BestTaste+.**

Recomendación:

- **Crear**: El usuario crea una recomendación. En esta función, el sistema le pide al usuario una serie de datos, los que incluyen la canción que desea recomendar, una sección de la canción a mostrar cómo preview con duración de 15 segundos, así como una descripción para la recomendación.
- **Borrar**: Se elimina la recomendación.
- **Canciones**:
- **Mostrar**: Enseña dicha lista de canciones.

Comentarios:

- **Crear**: El usuario crea un comentario bajo recomendación.
- **Borrar**: El usuario elimina el comentario.

Playlist:

- **Crear**: Se crea una playlist con un nombre que el usuario desee, y al crear se establece si se trata de una playlist pública o privada
- **Borrar**: elimina la playlist
- **Mostrar**: muestra las playlist del usuario
- **Mostrar playlist del sistema**: Lo mismo que el mostrar, solo que de una playlist creada por el sistema

Estadísticas:

- **Mostrar estadísticas**: Crea una lista de estadísticas de un usuario o canción según lo pedido por el usuario.

Administrador:

- **Dar de alta a sponsor**: Registrar un sponsor en la base de datos.
- **Dar de baja a sponsor**: Eliminar un sponsor
- **Mostrar**: Encargado de buscar y mostrar a los administradores.

1.2.3 Aspectos de rendimiento

Para lograr el máximo rendimiento, las funciones estarán relacionadas entre sí, además de optimizadas. Nos aseguraremos de que todas las acciones del usuario tengan un límite para así evitar sobrecargar el sistema.

Se escogerá un algoritmo eficiente para conseguir un gran rendimiento general, así como la

optimización del código, y se asignará un ID a cada usuario y a las canciones para ayudar con el rendimiento, y mantener un orden. Controlaremos la memoria para evitar fugas y optimizar el uso de recursos.

En cuanto a las actualizaciones futuras, estas no degradarán el rendimiento, lo mejorarán.

1.2.4 Restricciones y técnicas de gestión

Las restricciones que se nos puedan presentar a lo largo del proyecto quedarán más ampliamente especificadas en la SRS, sin embargo, incluimos un resumen.

Las principales son las restricciones de tiempo ya que solo contamos con tres meses para completar el proyecto, y la propia experiencia del personal, ya que este puede carecer de experiencia a la hora de usar Java, que es el lenguaje usado para desarrollar el código del programa, teniendo como consecuencia un código mal desarrollado.

Para gestionar estos riesgos se seguirá el mecanismo SQAS-SEI que nos ayudará a elaborar una tabla de prioridades y un plan RSGR para gestionar los riesgos.

1.3 Modelo de proceso

A la hora de escoger un modelo de proceso, es importante tener en cuenta todas las restricciones y recursos con los que contamos. Tenemos un hardware limitado, unos requisitos que pueden cambiar a lo largo del desarrollo del proyecto, entre otras cosas. Es por ello que necesitamos contar con un modelo que se adapte con facilidad a los cambios. Por ello hemos escogido un Proceso Unificado de Desarrollo o RUP (Rational Unified Process), ya que este se centra en la gestión de riesgos y en la producción de software de alta calidad, que es el objetivo de este proyecto.

Este modelo se caracteriza por ser iterativo e incremental, en estar dirigido por casos de uso, se centra en la arquitectura, es conceptualmente amplio y diverso con una constante evolución, adaptable y repetible, por lo que consideramos que es el más adecuado para nuestro proyecto software.

Por último, la estructura que utilizaremos en este proyecto será la Centralizada Controlada (CC) ya que somos un grupo pequeño de desarrolladores con tiempo limitado para desarrollar el software. Además, esta estructura funciona bien con el modelo de proceso escogido ya que el modelo es altamente racional y aborda la gestión de riesgos de manera integral, lo cual es coherente y beneficioso para una estructura que valora el control y la reducción de riesgos.

2. Estimaciones del proyecto

La presente estimación global de los diferentes parámetros del proyecto tiene por objetivo mostrar los resultados esperados en cuanto al uso de recursos financieros, energéticos, materiales y de personal. Lógicamente, la estimación no espera ser una guía infalible de cara a los futuros emplazamientos de material, recursos y costes, pero sí aspira a crear un marco de uso de estos lo más ajustado posible con el fin de guiar la planificación de tareas y objetivos de desarrollo.

Con este propósito en mente, lo que sigue es una visión completa de las diferentes estimaciones necesarias y ajustadas al proyecto a realizar.

2.1 Datos históricos

Hay que considerar que, debido a la reciente creación del grupo de desarrolladores, no se pueden deducir datos ni estadísticas de trabajos previos realizados conjuntamente por dicho grupo. Estos condicionantes, como es evidente, dificultan la creación de una perspectiva de planificación basada en experiencias previas en el entorno de este proyecto. Por lo tanto, consideramos que no es posible extraer datos relevantes de otras organizaciones y aplicaciones con los que incrementar la precisión de nuestra estimación.

Como consecuencia de esto, las únicas técnicas de estimación válidas son la descomposición en producto y el empleo de modelos paramétricos.

2.2 Técnicas de estimación

Para la estimación de este proyecto, usaremos las técnicas de descomposición. Estas técnicas son la descomposición basada en el proceso, y la descomposición basada en el problema.

En nuestro caso, no podremos realizar la estimación de nuestro proyecto con la descomposición basada en el problema, dado que esta necesita la existencia de datos históricos para calcular el tamaño del código y el esfuerzo que requiere el proyecto. Sin embargo, como ya se mencionó anteriormente, no contamos con dichos datos. Es por esto que proponemos realizar una descomposición basada en el proceso, construida a partir del análisis minucioso de cada módulo junto con sus respectivas funcionalidades.

2.2.1 Estimación por módulos

Para la descomposición siguiendo el criterio de división por módulos, se considerarán los ya mencionados de Usuario, Administrador, Comments, Estadísticas, Playlists, Sponsors y, por último, el módulo General, en el que se identifican los procesos necesarios para llevar a cabo las interacciones entre ellos, la interfaz de usuario y las bases de datos a las que recurre el sistema.

Lo que sigue a continuación en forma de tablas es una estimación del coste de cada módulo en función de las líneas de código (*LDC*, de ahora en adelante) necesarias para construir las funciones pertenecientes a cada uno.

MÓDULO USUARIO				
FUNCIÓN	$V_{optimista}$	$V_{probable}$	$V_{pesimista}$	$V_{esperado}$
Darse de alta	150	176	210	177
Darse de baja	110	135	155	134
Log in	50	68	75	66
Log out	45	60	85	61
Modificar la cuenta	20	30	38	29
Buscar canción	15	21	24	20
Encontrar canción favorita	38	50	65	50
TOTAL				537

MÓDULO ADMINISTRADOR				
FUNCIÓN	$V_{optimista}$	$V_{probable}$	$V_{pesimista}$	$V_{esperado}$
Acceder a la lista de usuarios	30	38	47	38
Dar de alta a sponsors	120	147	168	146
Dar de baja a sponsors	110	123	135	122
Dar de alta una canción	78	95	115	95
Dar de baja una canción	90	105	125	105
TOTAL				506

MÓDULO COMMENTS				
FUNCIÓN	$V_{optimista}$	$V_{probable}$	$V_{pesimista}$	$V_{esperado}$
Crear comentario	28	40	58	41
Reportar comentario	25	36	46	35
Realizar recomendación	35	45	52	44
Borrar recomendación	40	57	65	55
TOTAL				175

MÓDULO ESTADÍSTICAS				
FUNCIÓN	**$V_{optimista}$**	**$V_{probable}$**	**$V_{pesimista}$**	**$V_{esperado}$**
Mostrar estadísticas de un usuario	18	28	35	27
Mostrar estadísticas de una canción	26	35	43	34
TOTAL				61

MÓDULO PLAYLISTS				
FUNCIÓN	**$V_{optimista}$**	**$V_{probable}$**	**$V_{pesimista}$**	**$V_{esperado}$**
Acceder a playlist generada por el sistema	20	35	48	34
Crear playlist de canciones favoritas	50	78	95	76
Acceder a playlist de canciones favoritas	16	30	40	29
Editar playlist	32	50	65	49
TOTAL				188

MÓDULO SPONSORS				
FUNCIÓN	**$V_{optimista}$**	**$V_{probable}$**	**$V_{pesimista}$**	**$V_{esperado}$**
Dar de alta a sponsors	120	147	168	146
Dar de baja a sponsors	110	123	135	123
TOTAL				268

MÓDULO GENERAL				
FUNCIÓN	**$V_{optimista}$**	**$V_{probable}$**	**$V_{pesimista}$**	**$V_{esperado}$**
Diseñar esquemas UI	200	230	320	240
Configurar el arranque de la aplicación	60	82	95	80
Gestionar registros de comentarios	38	55	70	54
Programar transiciones entre pantallas de la UI	90	115	140	115
Resolver conflictos de llamadas a la Base de Datos	110	120	135	120
TOTAL				609

Puesto que, como se ha mencionado anteriormente, el equipo es de nueva formación y sus participantes nunca han realizado proyectos conjuntamente, no se dispone de una base de datos histórica de la cual poder extraer métricas sobre la productividad de la organización.

Teniendo esto en cuenta, se cree que una buena solución para esta carencia de información es considerar la productividad de cada programador, estando basada en el análisis de trabajos previos realizados individualmente por cada uno.

Según una disección de los mencionados proyectos en función de su duración, su complejidad en LDC y las circunstancias bajo las cuales fueron completados los últimos proyectos por parte de cada miembro del equipo, se ha llegado a la conclusión de que la media de productividad individual en la organización es de 170 LDC/M. Por lo tanto, haciendo una estimación conservadora –puesto que los proyectos grupales implican dificultades añadidas por cantidad mayor a la simple suma del rendimiento personal de las partes-, se considera que la organización debe ser capaz de alcanzar una productividad media de 1200 LDC/PM. La suma en LDC de los valores esperados para todos los módulos es 2344 LDC. Considerando un coste para la empresa de 6500 €/PM, se tienen los siguientes resultados:

- Esfuerzo = 2344 LDC / 1200 LDC/PM = 1,95 PM
- Coste = 1,95 PM * 6500 €/PM = 12675 €

2.3 Estimaciones de esfuerzo, coste y duración según diversos métodos de estimación paramétricos

Con el objetivo de incrementar la precisión y mejorar las aproximaciones hechas en esta estimación de proyecto, es fácil comprender la utilidad de emplear métodos de estimación paramétricos. De esta manera, se espera que un análisis basado en fórmulas obtenidas empíricamente dé una nueva perspectiva a las estimaciones basadas en el producto hechas anteriormente.

2.3.1 Ecuación del Software

Comenzaremos nuestro análisis con un clásico de las estimaciones paramétricas: la ecuación de Putnam. Esta fórmula fue introducida por Lawrence H. Putnam, originalmente en 1978. Tras analizar diversos proyectos software, este hombre dedujo una notable similitud entre la distribución de Rayleigh y la complejidad de los procesos de desarrollo software. De este estudio obtuvo una fórmula matemática, cuya expresión es la siguiente:

$$E = B \cdot \left(\frac{LDC}{P}\right)^3 \cdot \frac{1}{t^4}$$

Donde “E” es el esfuerzo medido en Personas-Mes (PM), “B” es un escalar que depende del tamaño del proyecto, “P” es la productividad de la organización y “t” es la duración del proyecto en meses. En el caso de ***BestTaste***, los parámetros tomarían como valores:

B = 0.26

P = 28000

t = 1.75

Con estos datos, según los cálculos de la ecuación de Putnam, nuestro proyecto tendría un coste de 0,16 PM

3. Estrategia de gestión del riesgo

Un riesgo es todo aquello que pueda afectar negativamente nuestro proyecto software. Existen tres tipos de riesgos que se nos puedan presentar a lo largo del desarrollo del proyecto, los riesgos del proyecto, los técnicos y/o los riesgos de negocio.

Para empezar nuestra gestión del riesgo, el primer paso es identificar los posibles riesgos distinguiendo entre los tipos mencionados anteriormente.

- Riesgos del proyecto:

  Entre estos, los riesgos posibles son:

  - Miembros: Esto se trata del riesgo de que haya falta de interés en el proyecto por parte de uno de los miembros del equipo. Esto supondría una carga de trabajo para el resto de los integrantes ya que al contar con un miembro menos en el equipo, se tendrán que repartir más tareas, afectando, así, el buen desarrollo del proyecto.

  - Recursos: Otro riesgo del proyecto es la posibilidad de que no dispongamos de las herramientas necesarias para desarrollar nuestro proyecto software. La falta de recursos puede suponer un proyecto sin acabar o mal hecho.

- Riesgos técnicos:

  Estos pueden ser:

  - Programas desconocidos: Uno de los riesgos técnicos más preocupantes es la falta de conocimiento en los programas usados por parte de algún miembro del equipo. Si alguien no sabe usar bien el software usado, podría llevar a retrasos en la entrega o, incluso, un proyecto mal desarrollado.

  - Diseño y definición de requisitos: A la hora de definir los requisitos, corremos el riesgo de que alguno de estos sea malinterpretado, y por ello, mal ejecutado a la hora de desarrollar el proyecto software. Es necesario una descripción clara y concisa de estos requisitos para evitar cualquier tipo de desarrollo inadecuado del proyecto.

  - Implementación del código: Si no se coordina bien la implementación del código, repartiendo bien las tareas y comprobando que avanza correctamente el desarrollo, tendremos un software mal diseñado.

3.1 Análisis del riesgo

Para asegurarnos que estudiamos bien los riesgos, se seguirá el mecanismo SQAS-SEI, el cual nos ayudará a desarrollar una tabla con nuestros propios riesgos para así poder analizarlos y agruparlos según la exposición al riesgo.

Prioridad: Tolerable (T), Bajo (L), Medio (M), Alto (H), Intolerable (IN)

Probabilidad: Frecuente, probable, ocasional, remota, improbable

Severidad: Catastrófica, crítica, severa, menor, irrelevante

Probability / Severity	Frequent	Probable	Occasional	Remote	Improbable
Catastrophic	IN	IN	IN	H	M
Critical	IN	IN	H	M	L
Serious	H	H	M	L	T
Minor	M	M	L	T	T
Negligible	M	L	T	T	T
LEGEND	T = Tolerable	L = Low	M = Medium	H = High	IN = Intolerable

Así, usando lo anterior, analizamos nuestros riesgos:

RIESGO	PROBABILIDAD	SEVERIDAD	PRIORIDAD
Miembros	Frecuente	Crítica	Intolerable (IN)
Recursos	Ocasional	Severa	Medio (M)
Programas desconocidos	Remota	Severa	Bajo (L)
Diseño y definición de requisitos	Ocasional	Crítica	Alto (H)
Implementación del código	Probable	Crítica	Intolerable (IN)

3.2 Estudio de los riesgos

Con el análisis de la exposición al riesgo, estudiamos los distintos riesgos presentes:

- Miembros: Este riesgo supone que el resto del equipo tenga más carga de trabajo que otros, resultando, a lo mejor, como demasiada carga para algunos miembros ya que, de por sí, todos cuentan con una carga elevada de trabajo. Tener que repartir trabajo por culpa de la falta de interés puede incluso llevar a falta de tiempo y, como consecuencia, no alcanzar el plazo de entrega estimado.
- Recursos: Puede que tengamos retrasos y dificultades en el transcurso del proyecto debido a la falta de disposición de ciertos materiales.
- Programas desconocidos: Si alguno de los miembros no dispone de los conocimientos necesarios para usar los programas, esto puede llevar a retrasos en

el desarrollo del código o del proyecto ya que tendrán que aprender a usarlo a base de prueba y error, en vez de poder desarrollar su parte de manera eficiente. Esto generaría, de la misma manera que algunos de los riesgos anteriores, retrasos en la entrega

- Diseño y definición de requisitos: El problema que se presenta con la definición de requisitos es que, si un requisito es malinterpretado por falta de explicación, puede llevar a un proyecto mal desarrollado y/o difícil de entender sobre todo para el cliente.
- Implementación del código: Si los miembros encargados del código no se coordinan bien entre sí puede ocurrir que hayan malentendidos entre estos, llevando a un código mal desarrollado ya que cada uno llevara a cabo su código con la idea que tiene en la cabeza, en vez de coordinar ideas con el compañero para que todo se complemente como debe. El resultado de esto es un código difícil de entender y muy mal hecho.

3.3 Plan de gestión del riesgo

El plan RSGR (Plan de Reducción Supervisión y Gestión del Riesgo) propone unos pasos a seguir para evitar que los riesgos anteriormente mencionados lleguen a ser un verdadero problema, y contar con *backups* en caso contrario.

Nosotros adoptaremos una estrategia proactiva por lo que desarrollaremos un plan de gestión para aquellos riesgos de prioridad intolerable, es decir, el riesgo de los miembros del equipo y el riesgo en la implementación del código.

- **Miembros**: El verdadero riesgo que corremos con la falta de interés de los miembros del equipo es que esto depende de las ganas que tengan de realizar su parte, y la actitud que adopten a la hora de afrontar el trabajo a realizar. Dependemos completamente del esfuerzo que esté dispuesto a poner cada persona y de su capacidad individual para afrontar los problemas. Es por esto que la solución que proponemos al problema es la motivación constante del equipo, sobre todo enfocada a los resultados que proporcionen los miembros, así reciben reafirmación positiva constante que les ayude a querer seguir trabajando.
- **Implementación del código**: La solución que proponemos para esto es que el jefe de equipo tenga máxima prioridad puesta en los miembros que desarrollan el código. El problema es que la programación es la parte más importante del proyecto ya que por mucho que tengas un proyecto bien definido, si luego el código no funciona, no sirvió de nada. Es por ello que creemos que mantener supervisión constante es la mejor solución.

4. Planificación temporal

Este apartado proporciona una ilustración sobre la organización que llevaremos a cabo para desarrollar el proyecto, teniendo como objetivo primordial evitar los retrasos en la entrega del software. El proyecto no solo se realiza con éxito cuando se entrega un producto satisfactorio para el cliente, también deben tenerse en cuenta otras restricciones como el coste y los plazos. Esta planificación implica asignar recursos, estimar la duración de las tareas definiendo la dependencia entre ellas y organizar la secuencia de actividades.

Es importante recordar que este es un primer planteamiento, en el cual, se realizan estimaciones detalladas, pero podrían surgir problemas o riesgos impredecibles que provocasen cambios en la planificación inicial.

4.1 Estructura de descomposición del trabajo/Planificación temporal

La descomposición del trabajo se realizará con el fin de asignar tareas de trabajo concretas, con plazos concretos a cada miembro del equipo. Para realizar una descomposición adecuada, basada en el producto, se lleva a cabo una partición horizontal, en la que descomponemos el sistema en módulos, es decir, en funciones que llevan a cabo tareas relacionadas o similares. A continuación, se realiza una partición vertical, descomponiendo los módulos en funciones o procesamientos directamente invocables por el usuario.

También es necesario distinguir actividades estructurales del proceso. Estas actividades deben descomponerse en acciones dependiendo del tipo de proyecto que se realiza.

A su vez, estas acciones pueden descomponerse nuevamente en conjuntos de tareas, con el fin de realizar un correcto desarrollo de las acciones.

4.2 Gráfico Gantt

4.3 Red de tareas

4.4 Tabla de uso de recursos

5. Recursos del proyecto

En este apartado desarrollaremos los recursos necesarios a lo largo del proyecto para la implementación de **BestTaste,** tanto en el uso del lenguaje de programación "Java", el personal, y el hardware y software.

5.1 Personal

- Teresa De La Vega: Experiencia de programación en C++ y java, coordinadora de proyecto.
- Alejandro Garrido: Experiencia de programación en C++ y java.
- Francisco Adrián Fonta: Experiencia de programación en C++ y java, diseñador de procesos y funciones.
- Álvaro Tejedor: Experiencia de programación en C++ y java, diseñador de procesos y funciones.
- Roi López: Experiencia de programación en C++, técnico de hardware y diseñador de interfaces gráficas.
- Jorge Martín: Experiencia de programación en C++ y java.
- Samuel Martínez: Experiencia de programación en C++ y java, diseñador.
- Jhon Fredy: Experiencia en C++, analista de proyecto.

5.2 Hardware y software

Los recursos que necesitaremos se diferencian entre los recursos Hardware y Software.

El Hardware se resume en los portátiles que necesitaremos cada uno para aumentar la eficiencia a la hora de trabajar.

Los recursos de Software que necesitaremos son aplicaciones como Eclipse, GitHub, IBM Rational Software Architect y Gmail.

- Eclipse: Se trata de una plataforma de software compuesto por un conjunto de herramientas de programación de código. La usaremos para la programación de **BestTaste** ya que, además, cuenta con una extensión "code together" que nos permite compartir en vivo el código, permitiéndonos desarrollarlo a la vez y corregir posibles errores que otros no sepan cómo solucionar en tiempo real.
- GitHub: Se trata de una herramienta online para compartición de código, que cuenta con un eficiente control de versiones que permite la gestión de grandes proyectos de forma sencilla y eficiente. De esta manera, no solo podremos controlar que todos estén completando su propia carga de trabajo, si no que podremos compartir las distintas versiones de manera eficaz a medida que avanza el proyecto.
- IBM Rational Software Architect: Este programa ayuda a reducir la complejidad de un proyecto para acelerar la innovación, mejorar la comunicación y la colaboración y facilitar el cumplimiento de dicho proyecto.

- Gmail: El cual usaremos para la comunicación entre integrantes del equipo.

5.3 Lista de recursos

- Jorge Martín.
- Teresa De La Vega.
- Alejandro Garrido.
- Francisco Adrián Fonta.
- Álvaro Tejedor.
- Roi López.
- Samuel Martínez.
- Jhon Fredy.
- Suscripción a Eclipse.
- Suscripción a code together.

6. Organización del personal

El desarrollo de una aplicación web exitoso implica una serie de cosas como, por ejemplo: una estructuración de equipos, una definición de roles y también de responsabilidades, así como la implementación de procesos para una comunicación efectiva y una toma de decisiones ágil. Esta organización es crucial para garantizar un desarrollo eficiente, minimizar los errores y asegurar una coordinación efectiva entre los miembros del equipo.

6.1 Estructura de equipo (si procede)

Durante el desarrollo de ***BestTaste*** usaremos una estructura de desarrollo centralizada controlada. Este enfoque nos garantizará una jerarquía clara haciendo pasar todas las decisiones por un líder definido, teniendo así mayor facilidad a la hora de tener que tomar decisiones de manera rápida y coherente, esta y muchas más como las siguientes son las razones que nos han llevado a escoger esta estructura:

1. Toma de decisiones centralizada: Todas y cada una de las decisiones que se tomen a lo largo del proyecto han de pasar por una misma figura siguiendo así un orden jerárquico que facilita y simplifica el proceso de desarrollo.
2. Estricta supervisión y control: Gracias a tener una autoridad que establece unas reglas y pautas a seguir, existe un alto nivel de supervisión que permite minimizar errores y unificar el proceso de desarrollo a uno solo.
3. Comunicación vertical: Imaginando la estructura del equipo como una pirámide, la comunicación se da desde la cúspide hacia los niveles inferiores y viceversa.
4. Mayor claridad sobre los roles y responsabilidades: Los roles y responsabilidades están claramente definidos, evitando así, confusiones y malentendidos sobre quién debe realizar cada tarea, facilitando así la asignación de responsabilidades y por lo tanto la gestión del proyecto.
5. Mayor capacidad de respuesta: Al tener una estructura centralizada controlada, el líder o el grupo de líderes puede responder rápidamente a los cambios o desafíos que puedan surgir durante el desarrollo del proyecto. Esto permite una adaptación ágil a las circunstancias cambiantes y una resolución rápida de problemas.

El líder de nuestro equipo encargado de la comunicación con el cliente y del reparto de tareas será Teresa De La Vega.

6.2 Informes de gestión

- **Reuniones fijas:** Llevaremos a cabo revisiones técnicas formales (RTF) con el equipo completo a través de reuniones semanales para evaluar el avance del proyecto. Estas reuniones también nos ayudarán a identificar posibles riesgos y asignar tareas para la próxima semana. Toda esta información será registrada en el informe de

gestión, y en cada reunión repasaremos lo que se haya documentado previamente para asegurarnos de que cada miembro haya cumplido con sus responsabilidades en el plazo establecido.

- **Reuniones Imprevistas:** Las reuniones improvisadas solo serán realizadas a consecuencia de grandes problemas encontrados en nuestro proyecto o si se requiere realizar un gran cambio exigido por el cliente. Estas reuniones las anunciará el jefe de equipo cierto día que pueda la mayoría del equipo. Si un miembro cree que es necesaria una reunión privada, primero notificará al jefe y éste avisará al resto para poner fecha.

En las reuniones imprevistas, no se llevarán a cabo las revisiones técnicas formales. En su lugar, se abordarán posibles soluciones o propuestas para los problemas identificados en el código o los cambios necesarios. No obstante, lo debatido en estas reuniones se documentará en el informe de gestión, ya que podría implicar cierto retraso o la dedicación de horas extras al proyecto.

7. Mecanismos de seguimiento y control

Utilizaremos aplicaciones como GoogleDrive y GitHub para gestionar los cambios del programa y los documentos destinados a la explicación del mismo.

Estas aplicaciones nos permitirán registrar el progreso de las actividades realizadas al momento sobre nuestro plan de proyecto. Destinaremos documentos para controlar el uso de las actividades del personal. Además, las aplicaciones nos ayudaran a identificar desviaciones o problemas a la hora de hacer cambios en el proyecto para, de esta forma, poder tomar medidas correctivas.

7.1 Garantía de calidad y control

Para conseguir una alta calidad del software se ha decidido implementar una secuencia de procedimientos que abarquen varias etapas del desarrollo del software y así garantizar un enfoque completo y estructurado del mismo.

1. Planificación y requisitos:

- Análisis de requisitos: Implica comprender las necesidades del solicitante del proyecto, por lo cual llevamos a cabo reuniones y discusiones para recopilar y documentar los requisitos. Estos requisitos, aunque no están definidos en un documento, fueron tratados con el solicitante del proyecto (el profesor) en clases prácticas y tutorías.

- Planificación de calidad: También definiremos criterios que servirán como referencia para evaluar el proyecto. Y estableceremos procesos para controlar la calidad en cada etapa del desarrollo. (por ejemplo, el documento de control de cambios).

2. Diseño:

Para clarificar los requisitos utilizamos diagramas de casos de uso y tablas explicativas. Todo esto siendo enfocado en la funcionalidad y eficiencia del producto.

3. Desarrollo:

El proyecto estará susceptible a los cambios si al realizar pruebas unitarias, pruebas de integración y pruebas funcionales, obtenemos una mejora en la colaboración del equipo y/o un enfoque que nos permita facilitar pruebas y correcciones del proyecto.

- Pruebas unitarias: Verifica el funcionamiento individual de cada componente.

- Pruebas de integración: Asegura que los módulos funcionen juntos.

- Pruebas funcionales: evalúan todo el proyecto para garantizar que cumple los requisitos.

4. Gestión y control de cambios.

Está etapa ha sido extensamente explicada en el apartado 7.2.

5. Mantenimiento:

Debido a que el proyecto aún se encuentra es sus fases iniciales, no es posible establecer un plan para corregir errores, actualizar y mejorar el software; sin embargo, se consideran oportuno mantener esta etapa vigente, dado que en un futuro será muy necesaria para la recolección de información de los usuarios la cual ayuda a identificar mejoras y futuras actualizaciones del software.

6. Gestión de calidad:

Es imprescindible realizar revisiones periódicas del proyecto, las cuales se prevén ayuden a identificar problemas o encontrar mejoras del mismo. Entre dichas revisiones tenemos la revisión de seguridad, la cual comprueba la seguridad del software, la revisión de calidad, en la que esperamos asegurar que el software funcione en condiciones perfectas y, por último, revisión de fallos y mejoras del producto.

Estos procedimientos están previstos para implementarse en cada fase del ciclo de vida del desarrollo del software.

A su vez, es importante declarar distintos grupos de personas para la realización de tareas de estos procedimientos, especialmente en las revisiones las cuales ocuparan gran parte del tiempo de trabajo. Nuestro equipo de proyecto actualmente está formado por 8 integrantes, por lo cual, al ser tan pocos, hemos estado tomando las decisiones del proyecto entre todos, aunque dependiendo del líder de proyecto, que en este caso es Teresa De La Vega. Prevemos para un futuro pronto repartir dichas tareas.

7.2 Gestión y control de cambios

El proyecto contiene un gran número de artefactos (del tipo diseños, programas, pruebas, planificaciones, etc..), para los cuales es necesario establecer una gestión eficiente de sus cambios. Por tanto, es importante analizar los cambios antes de realizarlos, registrarlos antes de implementarlos en el proyecto oficial, comunicarlos con los integrantes del proyecto o las personas encargadas de resolver dicho cambio y controlarlos de forma que mejoren la calidad y reduzcan los errores.

La gestión de configuración del software la tenemos en cuenta desde la etapa inicial del proyecto, en la cual podemos identificar de momento 3 artefactos, El Plan de Proyecto, La Especificación de Requerimientos del Software (SRS) y El Control de Cambios. Aparte de estos tres, existen otros artefactos no tan notorios como el diagrama de casos de uso, módulos de usuarios, gráfica de Gantt, los cuales también son importantes para el desarrollo y futuros mantenimientos del sistema.

Definimos un protocolo para el nombramiento de los elementos de configuración software, dando nombres que reflejan claramente el propósito del elemento. Es un nombre simple, para no crear confusiones, y un identificador de la versión, un ejemplo puede ser:

- Nombre del módulo: "PlanDelProyecto".

- Versión: "v1.2.0".

- Formato completo: "PlanDelProyecto_v1.2.0".

La persona Jhon Fredy Jaramillo López estará encargada de ser responsable de la Gestión de cambios y Creación de Líneas base.

Gestión de cambios:

1. Evaluar solicitudes de cambio recibidas.
2. Aprobar o rechazar las solicitudes de cambio y priorizar cambios en función de su urgencia e impacto.
3. Mantener un registro y seguimiento detallado de todos los cambios propuestos, aprobados, implementados y rechazados.
4. Informar a los miembros del equipo sobre los cambios aprobados y su nuevo resultado en el proyecto.
5. Asegurar que los cambios estén documentados y concuerden con las diferentes versiones del software.

Creación de líneas base:

1. Identifica los elementos que componen una versión estable y funcional del software.
2. Establece una versión del software como línea base, resaltándola para diferenciarla de otras versiones.
3. Gestiona los cambios de líneas base. Documentando dichos cambios y confirmando que estén previamente autorizados.
4. Realiza el seguimiento de las líneas base para mantener concordancia entre las versiones.

Sistema de control de versiones: Como hemos dicho anteriormente, utilizaremos aplicaciones como GoogleDrive y GitHub para rastrear y gestionar cambios del proyecto. Además, utilizaremos documentos diferentes para gestionar nuevas características o solucionar problemas sin afectar los documentos principales. A su vez, debemos constatar el uso de diferentes aplicaciones como IBM RSAD en la cual se ha utilizado para realizar diagramas como el de Casos de uso. Todo esto estará un poco más detallado en la base de datos de GCS.

Gestión de cambios: Cuando debemos solicitar un cambio en el documento, tenemos que incluir la documentación detallada del cambio, para que de esta forma el equipo encargado de la gestión de cambios pueda dar su aprobación.

Cabe destacar que la gestión de cambios y la gestión de versiones se harán en GoogleDrive.

8. Apéndices