BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

INGENIERÍA DE SOFTWARE

DRA. ETELVINA ARCHUNDIA SIERRA

DOCUMENTO DEL PROYECTO: NOCIONES BÁSICAS DE LA ING DE SOFTWARE.

ALUMNOS:

BARBOSA CARRILO, ANTONIO A

ESPINOSA VELAZQUEZ, LUIS A

MATEOS ROMERO, PEDRO

NORIEGA SILVESTRE, JESUS A

RUBIO MARTINEZ, JEDUS D

TALAVERA SANDOVAL, IVÁN

OTOÑO 2021

30-08-2021

INDICE

HOJA DE CONTROL DE CAMBIOS

FECHA	CAMBIOS
Semana 1	Se realizo la importancia de la ingeniería de software y tipos de software, se realizó la conclusión.
Semana 2	Se realizo la ética del desarrollo de software, se actualizo la conclusión.
Semana 3	Se realizaron los roles del equipo, se actualizo la conclusión.
Semana 4	La realizo la gestión y estimación de proyecto, se actualizo la conclusión.
Semana 5	Se realizo la investigación de modelos de software, se actualizo la conclusión.
Semana 6	Se realizo la ingeniería de requerimientos, se actualizo la conclusión.
Semana 7	Se realizaron los requerimientos del proyecto (Anexo 1), se actualizo la conclusión.
Semana 8	Se desarrollo el diseño del proyecto en base a el uml de usos y
Semana 9	requerimientos, así como una síntesis para la especificación (Anexo 2), se actualizo la conclusión.
Semana 10	Se desarrollo la síntesis de patones de diseño. Anexo 3, Patrones de
Semana 11	diseño y pseudocódigo del programa, se actualizo la conclusión
Semana 12	Investigación y desarrollo de patrones de diseños: decoradora, fabrica,
Semana 13	fábrica de métodos, actualización de la conclusión
Semana 14	Prototipo funcional del proyecto, anexo 4 repositorio con el prototipo
Semana 15	Implementación del proyecto, calidad, verificación y validación del software
Semana 16	Mantenimiento necesario para el software creado así como los tipos de mantenimiento

IMPORTANCIA DE LA INGENIERÍA DE SOFTWARE.

Actualmente vivimos en un mundo en el que cada vez más tareas se realizan a través de algún tipo de software y en el que todos los problemas tienen al menos un elemento que se debe desarrollar a través de una máquina. Tal es su importancia en el mundo moderno que toda la economía mundial depende del software, pero ¿cuál es la razón detrás de esta dependencia? Bueno, para entender esto es necesario tener una noción general de qué es la ingeniería de software y un poco de su historia.

El termino de "ingeniería de software" no tiene una fecha de aparición precisa ya que algunas personas acuñan el término a Friedrich Bauer, quien dio una conferencia de ingeniería de software de la OTAN en 1968, sin embargo, también lo usó Anthony Oettinger en 1966 hablando a cerca de las diferencias entre sistemas de software y ciencias de la computación y, por último, se encuentra Margaret Hamilton, quien usó el término mientras trabajaba en el programa espacial Apollo entre 1963 y 1964. Teniendo este periodo de tiempo en mente, uno de los factores que impulsó el desarrollo acelerado de esta disciplina fue la crisis de software que sucedió entre 1960 y 1980, llegando a cambiar metodologías y procesos del mantenimiento de sistemas informáticos. Esta famosa crisis del software (que muchos atribuían a la falta de compromiso por parte de los programadores para entregar herramientas eficaces y que cumplieran con las especificaciones dadas por el cliente) fue un importante tema de análisis para empresas y desarrolladores, teniendo que establecer una profesión formal para este campo de estudio.

Y posteriormente, con la aparición del internet, la demanda de herramientas para desplegar información, traducir sus flujos a otros idiomas, manejar ilustraciones, mapas, animaciones, etc. hizo que el crecimiento en esta área fuera explosivo. Entre 1970 y 1990 la ingeniería de software contribuyó con aproximadamente 90,000 millones de dólares al año, pero también surgieron múltiples virus de computadora, así como problemas de almacenamiento de información. Así, con el paso de los años mientras más crecía la demanda de software, más altos eran los estándares de calidad y se requería que se desarrollara en el menor tiempo posible.

Una vez teniendo este contexto, se puede proceder a un análisis más preciso: la ingeniería de software es una disciplina que se centra en el desarrollo de programas informáticos para la resolución de problemas específicos que varían en tamaño, dificultad y tipo. Sin embargo, no es tan simple como parece ya que este proceso de creación de un proyecto de ingeniería de software es muy amplio, abarcando desde el análisis previo del problema hasta la última fase de desarrollo de la solución. En palabras de Bauer (1972): "*La ingeniería de software trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales"*.

Así pues, esta rama de la ciencia de la computación contempla todas las fases del ciclo de vida de los sistemas informáticos, ciclo que se divide en diferentes fases y que cambia dependiendo del tipo de proyecto que se esté desarrollando y de muchos otros factores tales como la organización, los requerimientos administrativos, el tiempo previsto para la ejecución, etc. Además, tiene un enfoque de calidad, procesos, métodos y herramientas que permiten la correcta gestión y planeación de los proyectos. Debido a esta gran variedad de tareas por cubrir, es necesario repartir las actividades que se desarrollan en cada una de las fases del ciclo de vida del software. En general, estos roles son los siguientes:

-Análisis

-Diseño

-Programación

-Testeo

-Pruebas de calidad

-Administración de configuración.

-Validación y verificación

-Documentador

-Manutención

Todo esto con el fin de alcanzar todos los objetivos y especificaciones del proyecto y que en general, el producto final debe ser muy eficaz, seguro, adaptable a nuevas tecnologías y lo más importante, aceptable y de agrado para los usuarios que lo ocupen. Además, se deben enfrentar de manera ágil todos los problemas que se presenten, en especial en el ámbito de seguridad y de evolución social, ya que esto podría llevar el proyecto a un periodo de vida corto.

Además, contrario a lo que se cree, el desarrollo de software no es tan flexible ni tan sencillo de modificar, esto es solo uno de los mitos más comunes que existen en este ámbito. De forma general, se puede decir que existen tres clases de mitos:

Gestión:

- Manuales ya hechos para el desarrollo de software.
- Se pueden agregar programadores para terminar más rápido el proyecto.
- Solo son necesarios objetivos generales.

Clientes:

- El software es flexible y se pueden añadir cambios de manera rápida.
- El trabajo termina con la finalización de la herramienta y su entrega.

Desarrolladores:

- Solo se puede verificar la calidad con el producto terminado.
- Solo es necesario entregar el programa.

Aunque no lo parezca, los mitos de clientes también nos involucran a nosotros y tienen una estrecha relación con el planteamiento del proyecto así como de la estimación de tiempo que hagamos, ya que en las reuniones que tengamos con el cliente, se le debe dejar en claro que el tiempo calculado por nosotros se debe respetar y que no es posible agregar al equipo más personas para terminar antes de lo previsto, ya que esto más que hacernos adelantar el trabajo, hará que tengamos que pausar el desarrollo para poner al tanto a los nuevos integrantes y para volver a repartir las tareas, además de que el ritmo de trabajo será notoriamente inferior gracias a que los nuevos integrantes no estarían familiarizados con el proyecto .

Como se puede ver, la planificación detallada es un paso fundamental en el desarrollo de software, ya que no solo implica programar. Hay actividades muy diversas que requieren muchas áreas del conocimiento y que son vitales para no cometer los errores plasmados en los mitos. Por esto mismo, los proyectos desarrollados deben ser hechos con detalle, con un tiempo establecido, con compañeros específicos y atendiendo siempre las necesidades del cliente, siendo claros y directos con los tiempos de desarrollo. Todo esto para generar productos con los más altos estándares de calidad.

TIPOS DE SOFTWARE

Se conoce como software de sistema o software de base a la serie de programas preinstalados en el computador o sistema informático y que permiten interactuar con el Sistema Operativo (el software que rige el funcionamiento del sistema todo y garantiza su operatividad), para dar soporte a otros programas y garantizar el control digital del hardware. De ahí la importancia y pertinencia de conocer estos.

Algunos ejemplos de software del sistema serían: Cargadores de programas (loaders), Sistemas operativos, programas utiliarios básicos, BIOS, Líneas de comandos.

Ejemplos de software de programación son: compiladores y editores de texto.

Ejemplos de software de aplicación: navegadores, reproductores de video, antivirus etc.

Existen diversos tipos de software entre los que se destacan:

-Software genérico.

-Software hecho a la medida.

Software genérico:

El software genérico son sistemas que son diseñados para un mercado en específico y son comprados por cualquier persona u organización que estén interesados en comprarlos, ejemplos de software genéricos determina la especificación del producto qué es lo que debe hacer y cómo debe funcionar.

Software hecho a la medida:

El software hecho a la medida es software mandado a hacer para un cliente en particular que resuelva un problema en específico, por ejemplo: un sistema de facturación que pueda utilizarse a través de todos los países del mundo y pueda manejar los diferentes tipos de códigos impuestos a través de todas las jurisdicciones. Con el software hecho la medida las especificaciones entre otras decisiones del producto son determinadas por el cliente.

En el contexto de la Ingeniería del Software, un proceso no es una prescripción rígida de cómo se construye un sistema software. Como hay muchos tipos diferentes de software, no existe un proceso software universal. No obstante, cualquier proceso debe incluir, de alguna manera, las cuatro actividades principales de la Ingeniería del Software:

1. Especificación del software. Dónde se definen la funcionalidad del software y sus restricciones.

2. Desarrollo del software. Se produce el software que cumple con las especificaciones.

3. Validación del software. Se debe asegurar que el software cumple con lo que el cliente espera.

4. Evolución del software. El software debe evolucionar para cumplir con las necesidades cambiantes del cliente.

Es de vital importancia tener muy presentes y conocer los diferentes tipos de software que existen, ya que de esto depende las estrategias que usemos para abordar ciertos problemas que tengamos en el proceso de desarrollo, así como el planteamiento de objetivos, el enfoque que se le dará, los alcances y limitaciones que tendrá. Además, conocer con certeza el tipo de software que se va a desarrollar permite un trabajo eficiente y centrado en las características específicas de este. Cabe destacar que nosotros como desarrolladores y conocedores del tema, debemos ser claros a la hora de tratar con nuestros clientes, ya que ellos desconocen toda esta información, por lo que es probable que no sea de su conocimiento qué tipo de software específico necesitan

para el problema que quieren resolver, por lo que dependiendo de las características y funciones que soliciten , nosotros deberemos indicar qué tipo de software será; cuáles van a ser sus funciones, qué se requiere para su correcta implementación y cuáles serán los alcances que se tendrán.

La proyección que tenemos en nuestro equipo si bien no podría afirmarse que es para un beneficio a grandes escalas, ciertamente resultaría beneficioso no sólo para nosotros, sino para nuestros compañeros, dado que facilitaría en cierta medida la vida estudiantil y de cierta forma problemas consecutivos a este. Dado que dentro de la comunidad universitaria se usaría de forma general, dicho de otra forma, software genérico y de aplicación usual. Ya que nuestro proyecto no atañe al **funcionamiento** del computador, sino que será instalado en el sistema para conseguir una función determinada, con el propósito descrito anteriormente.

ÉTICA DEL DESARROLLO DE SOFTWARE.

Los desarrolladores de software cargan con un pero ético muy importante que es necesario que siempre tengan en mente, ya que, dependiendo del enfoque del proyecto, se podría llegar a dañar al cliente, a los usuarios o a bienes materiales ajenos. Por esto mismo, es vital analizar con cuidado todos los aspectos que puedan generar problemas de ética o incluso legales (como los derechos de propiedad), gracias a esto, sociedades de los estados unidos cooperaron para producir un código de ética práctico, que contiene ocho puntos principales los cuales toman en cuenta a no solo a la sociedad si no también el desarrollo, el producto y el ambiente laboral los cuales se desglosan de manera más amplia a continuación

SOCIEDAD

Con respecto a la sociedad el ingeniero en software tiene la obligación de actuar de manera coherente con el interés social, aceptar las responsabilidades por el trabajo realizado, deberá ser mediador en los interese del ingeniero, empresario, el cliente y los usuarios con el fin del bienestar publico

Deberá dar su aprobación al software desarrollo solo si tiene una creencia funda en hechos de que es seguro, cumple con las especificaciones, ha pasado por pruebas estrictas y pertinentes, así como no disminuye la calidad de vida, daña la confidencialidad de los usuarios o daña de alguna manera el medio ambiente

De igual manera debe mostrar a las personas o autoridades correspondientes cualquier peligro que viole algún derecho del usuario, la sociedad o el medio ambiente que sea considerado esté relacionado con el software desarrollado.

CLIENTE Y EMPRESARIO

La ética entere el cliente o empresario y el ingeniero en software consiste en que este debe actuar de manera que produzca el mejor resultado para quien labore por lo que no deberá aceptar trabajos externos que pudieran afectar u obstaculizar el desarrollo del proyecto principal, el ingeniero en software está en la obligación de proporcionar servicios solo en las áreas de su competencia, debe ser honesto y sincero acerca de sus limitaciones en su experiencia o educación.

No deberá utilizar software de índole ilegal de manera consiente, así como la propiedad del cliente solo deberá ser utilizada bajo su consentimiento y supervisión

Cualquier información obtenida mediante el trabajo profesional deberá ser privada siempre que esta no atente con aspectos generales de interés o la ley

Se deberá identificar, recabar, documentar evidencias e informar con antelación al cliente si el proyecto corre el riesgo de fracasar, que sobre pase el límite de capital o viole leyes con respecto a la propiedad intelectual

PRODUCTO

La ética al realizar un producto es importante ya que debe garantizar que los productos y las modificaciones de estos cumplan con los más altos estándares profesionales

Al momento del desarrollo del producto se deben garantizar objetivos adecuados y alcanzables, así como debe proveer la máxima calidad, coste y realización en un plazo aceptables. Una parte importante al momento del desarrollo es Identificar, definir y examinar temas éticos económicos, culturales, legales y medioambientales relacionados con el proyecto al igual que seguir una metodología adecuada teniendo en cuenta la investigación previa del impacto del producto en cuestión

VALORACIÓN

En la ética de valoración deberá comparar la viabilidad al tomar una decisión sin poner de lado el beneficio colectivo, así como el beneficio personal ya que esta puede traer consecuencias para el proyecto estas pueden abarcar desde la evaluación de cualquier software o documento, no involucrarse en prácticas financieras engañosas como sobornos, dobles facturaciones etc. así como firmar documentos preparados bajo su supervisión o dentro de sus áreas de competencia y con los que está de acuerdo

GESTIÓN

En la gestión del proyecto es bastante importante la ética puesto que es donde se recluta al personal de trabajo los cuales deben ser atraídos al proyecto otorgando una descripción completa y precisa de las condiciones del empleo, así como informar sobre la remuneración de su trabajo la cual debe ser justa, respecto a la asignación de roles

para cada miembro no debe existir preferencias al momento de asignación ya que esto dependerá únicamente de la capacidad y compromiso que tenga el mimbro.

Es importante recordar que en esta parte es donde se platea el desarrollo el cual debe ser de calidad y reduzca el mayor número de riesgos a su vez debe garantizar estimaciones cuantitativas realistas del coste, plazo, personales, calidad y productos

PROFESIÓN

La función de la ética en la profesión es avanzar en la integridad y reputación de la profesión de manera consistente con el interés social, así como ayudar al desarrollar un ambiente organizativo y un comportamiento ético

De igual manera promover el conocimiento general de la ingeniería del software, así como diseminar el conocimiento mediante la participación en organizaciones profesionales, reuniones y publicaciones. Se debe de ser preciso en la descripción de las características del software en el que se trabaja, evitando no solo falsas declaraciones, sino también declaraciones que podrían razonablemente suponerse especulativas, vacías, decepcionantes, engañosas o dudosas Tener la responsabilidad de detectar, corregir e informar errores en el software y documentos asociados en los que se trabaje así que el ingeniero en software no promoverá el interés propio a costa de la profesión.

COMPAÑEROS

Serán justo y apoyarán a sus compañeros esto se puede lograr animando a los compañeros a adherirse a este código, ayudar a los compañeros en el desarrollo profesional, reconocer completamente el trabajo de otros y sin atribuirse méritos que nos son propios, revisar el trabajo de otros de manera objetiva, sincera y adecuadamente documentada, tratar justamente las opiniones, preocupaciones o quejas de un compañero

Debemos apoyar a los compañeros en la comprensión sobre los estándares de trabajo, incluyendo políticas y procedimientos para proteger las claves de acceso, ficheros y otras información confidencial y medidas de seguridad en general

No interferir injustamente en la carrera profesional de cualquier compañero, sin embargo, la preocupación por el empresario, el cliente o el interés público puede forzar, con buena voluntad, a cuestionar las competencias de un compañero

PERSONAL

El compromiso ético personal se basa en que se participara en el aprendizaje continuo referente a la práctica de su profesión y promoverán un enfoque ético en este

Deberá mejorar su conocimiento de los avances en los análisis específicos, diseño, desarrollo, mantenimiento y pruebas del software y documentos relacionados, junto con la gestión del proceso de desarrollo, deberá mejorar produciendo documentación precisa informativa y correctamente escrita, mejorara su compresión de software y documentos realizados en los que trabaja y del entorno que utilizara, debe mejorar sus conocimientos de los estándares pertinentes y de las leyes que regulan el software

LOS ROLES DEL EQUIPO, TOMANDO COMO BASE EL CICLO DE VIDA DE SOFTWARE.

El ciclo de vida del desarrollo del software contempla las fases necesarias para validar el desarrollo del software y así garantizar que este cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo, asegurándose de que los métodos usados son apropiados.

El desarrollo de software es una actividad que, dada su complejidad, debe desarrollarse en grupo. Además, esta actividad requiere de distintas capacidades, las que no se encuentran todas en una sola persona. Por ello, se hace necesario formar el grupo de desarrollo con las personas que cubran todas las capacidades requeridas. Cada una de esas personas aportará al grupo parte del total de las capacidades necesarias para llevar a cabo con éxito el desarrollo. Por ello, es que cada persona debe tener un rol dentro del grupo, que viene dado por su experiencia y capacidades personales. Estos roles son, administrador de proyecto, analista, diseñador, programador, tester, asegurador de calidad, documentador, ingeniero de manutención, ingeniero de validación y verificación, administrador de la configuración y, por último, el cliente. Para cada uno de estos roles, se definen sus objetivos, actividades, interacción con otros roles, herramientas a utilizar, perfil de las personas en ese rol y un plan de trabajo. Hay que señalar que es posible que no se requieran todos los roles en un desarrollo

FASES DE DESARROLLO DE SOFTWARE

La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con grandes posibilidades de éxito. Esta sistematización indica cómo se divide un proyecto en módulos más pequeños para normalizar cómo se administra el mismo.

De esta forma, las etapas del desarrollo de software son las siguientes:

PLANIFICACIÓN

Antes de empezar un proyecto de desarrollo de un sistema de información, es necesario hacer ciertas tareas que influirán decisivamente en el éxito del mismo. Algunas de las tareas de esta fase incluyen actividades como la determinación del ámbito del proyecto, la realización de un estudio de viabilidad, el análisis de los riesgos asociados, la

estimación del coste del proyecto, su planificación temporal y la asignación de recursos a las diferentes etapas del proyecto.

En esta fase se encarga el administrador de proyectos

ANÁLISIS

Hay que averiguar qué es exactamente lo que tiene que hacer el software. Por eso, la etapa de análisis en el ciclo de vida del software corresponde al proceso a través del cual se intenta descubrir qué es lo que realmente se necesita y se llega a una comprensión adecuada de los requerimientos del sistema

En esta fase se encarga el administrador de proyectos, el analista y el documentador

DISEÑO

En esta fase se estudian posibles opciones de implementación para el software que hay que construir, así como decidir la estructura general del mismo. El diseño es una etapa compleja y su proceso debe realizarse de manera iterativa.

En esta fase se encarga el diseñador y el documentador

IMPLEMENTACIÓN

En esta fase hay que elegir las herramientas adecuadas, un entorno de desarrollo que facilite el trabajo y un lenguaje de programación apropiado para el tipo de software a construir. Esta elección dependerá tanto de las decisiones de diseño tomadas como del entorno en el que el software deba funcionar.

En esta fase se encargan el diseñador y los programadores

PRUEBAS

la fase de pruebas del ciclo de vida del software busca detectar los fallos cometidos en las etapas anteriores para corregirlos.

En esta fase se encargan los tester y el documentador

INSTALACIÓN O DESPLIEGUE

La siguiente fase es poner el software en funcionamiento, por lo que hay que planificar el entorno teniendo en cuenta las dependencias existentes entre los diferentes componentes del mismo.

Es posible que haya componentes que funcionen correctamente por separado, pero que al combinarlos provoquen problemas. Por ello, hay que usar combinaciones conocidas que no causen problemas de compatibilidad.

En esta fase se encargan los programadores

USO Y MANTENIMIENTO

Esta es una de las fases más importantes del ciclo de vida de desarrollo del software. Puesto que el software ni se rompe ni se desgasta con el uso su mantenimiento incluye tres puntos diferenciados:

- Eliminar los defectos detectados durante su vida útil (mantenimiento correctivo).
- Adaptarlo a nuevas necesidades (mantenimiento adaptativo).
- Añadirle nuevas funcionalidades (mantenimiento perfectivo).

En esta fase se encargan los programadores y los documentadores

GESTIÓN Y ESTIMACIÓN DE PROYECTO

La gestión de nuestro proyecto empieza con el alcance del mismo, las gestiones que más utilizaremos en este caso son la de recursos humanos, tiempo, los interesados, calidad, los riesgos y la comunicación.

Teniendo como recursos a nosotros mismos como equipo con su respectiva computadora e internet y contando como tiempo límite lo que resta del semestre.

Si se trata de los interesados, es en cuestión de a quienes en el circulo específico en el que estará el programa lo ocuparán y en el mejor de los casos, lo recomendarán una vez visto el resultado esperado. En cuanto a riesgos, se trata de evitar el retraso en la programación, algún bug o problema grave que altere el funcionamiento del programa para el usuario, mientras que a calidad se refiere, se trata de poner el mejor de los esfuerzos logrando que el programa funcione evitando los riesgos antes mencionados antes y después a manera de mantenimiento.

Por último, en la comunicación por nuestra parte, tener siempre una disposición para estar al pendiente en cada parte del proyecto en cualquier asunto, una vez listo el proyecto, mantener una comunicación con los usuarios a manera de feedback, creando un entorno donde el los sean los que mejoran el programa con sus sugerencias.

Estimación de software por puntos de función:

Generador de horarios (funcionalidades):

- *Ingresar nuevos horarios*
- *Modificar horarios existentes*
- *Generar una lista con los horarios inscritos*

Componente	*Tipo de componente*
Ingreso de horarios	*Entrada externa*
Modificar horarios	*Entrada externa*
Generar lista de horarios	*Salida externa*
Tabla de horarios guardados	*Archivo lógico interno*

Componente	**Tipo de componente**	**Nivel de complejidad**	**Puntos de función**
Ingreso de horarios	Entrada externa	Bajo	3
Modificación de horario	Entrada externa	Medio	5
Listado de horarios	Salida externa	Bajo	3
Tabla de horarios guardados	Archivo lógico interno	Medio	7

MODELOS DE PROCESOS DE SOFTWARE

Los procesos de software son las actividades para diseñar, implementar y probar un sistema de software. El proceso de desarrollo de software es complicado e implica mucho más que conocimientos técnicos.

Ahí es donde los modelos de procesos de software son útiles. Un modelo de proceso de software es una **representación abstracta** del proceso de desarrollo.

Un modelo definirá lo siguiente:

- Las tareas para realizar
- La entrada y salida de cada tarea.
- Las condiciones previas y posteriores para cada tarea
- El flujo y la secuencia de cada tarea.

Hay muchos tipos de modelos de proceso para satisfacer diferentes requisitos. Nos referimos a estos como modelos SDLC (modelos de ciclo de vida de desarrollo de software). Los modelos SDLC más populares e importantes son los siguientes:

MODELO DE CASCADA

El desarrollo en cascada (en inglés, waterfall model) es un **procedimiento lineal** que se caracteriza por dividir los procesos de desarrollo en sucesivas fases de proyecto. Al contrario que en los modelos iterativos, cada una de estas fases se ejecuta tan solo una vez. Los resultados de cada una de las fases sirven como hipótesis de partida para la siguiente. El waterfall model se utiliza, especialmente, en el desarrollo de software.

El desarrollo del modelo se atribuye al teórico de la informática Winston W. Royce. Sin embargo, Royce no es el inventor de este modelo. Muy al contrario, en su ensayo de 1970 titulado *Managing the Development of Large Software Systems,* el teórico presenta una **reflexión crítica acerca de los procedimientos lineales**. A modo de alternativa, Royce presenta un modelo iterativo incremental en el que cada una de las fases se basa en la anterior y verifica los resultados de esta.

Royce propone un modelo compuesto por siete fases que se ha de ejecutar en diversas vueltas (iteraciones):

1. **Requisitos de sistema**
2. **Requisitos de software**
3. **Análisis**
4. **Diseño**
5. **Implementación**
6. **Prueba**
7. **Servicio**

El procedimiento popularmente conocido como waterfall model se basa en las fases definidas por Royce, **pero solo prevé una iteración.**

En el ensayo publicado por Royce, el término no aparece en ningún momento.

En la práctica, se aplican **diversas versiones del modelo**. Los más habituales son los modelos que dividen los procesos de desarrollo en cinco fases. En ocasiones, las fases 1, 2 y 3 definidas por Royce se integran en una sola fase de proyecto a modo de análisis de los requisitos.

8. **Análisis:** planificación, análisis y especificación de los requisitos.
9. **Diseño:** diseño y especificación del sistema.
10. **Implementación:** programación y pruebas unitarias.
11. **Verificación:** integración de sistemas, pruebas de sistema y de integración.
12. **Mantenimiento:** entrega, mantenimiento y mejora.

La siguiente imagen explica por qué el procedimiento lineal se denomina metodología en cascada.

El modelo en cascada de cinco niveles, basado en las propuestas de Winston W. Royce, divide los procesos de desarrollo en las siguientes fases de proyecto: análisis, diseño, implementación, verificación y mantenimiento. El gráfico incluye una de las ampliaciones del modelo planteadas por Royce: la verificación de los resultados de cada una de las fases tomando en consideración las exigencias y especificaciones formuladas en el paso anterior.

En las ampliaciones de la metodología en cascada se añaden funciones iterativas al modelo básico como, por ejemplo, los saltos hacia atrás, que permiten comparar los resultados de cada una de las fases con las hipótesis obtenidas en la fase anterior, de modo que se puedan verificar.

Las fases del desarrollo en cascada

En este modelo, las diferentes fases de un proceso de desarrollo se suceden una detrás de otra como en una cascada. Cada una de las fases concluye con un **resultado provisional (hito)** como, por ejemplo, un catálogo de requisitos en forma de pliego de condiciones, la especificación de una arquitectura de software o una aplicación a nivel alfa o beta.

Análisis

Todo proyecto de software comienza con una fase de análisis que incluye un estudio de viabilidad y una definición de los requisitos. En el **estudio de viabilidad** se evalúan los costes, la rentabilidad y la factibilidad del proyecto de software. El estudio de viabilidad da como resultado un pliego de condiciones (una descripción general de los requisitos), un plan y una estimación financiera del proyecto, así como una propuesta para el cliente, si fuera necesario.

A continuación, se realiza una **definición detallada de los requisitos**, incluyendo un análisis de la situación de salida y un concepto. Mientras que los análisis de salida se encargan de describir la problemática en sí, el concepto ha de definir qué funciones y características debe ofrecer el producto de software para cumplir con las correspondientes exigencias. La definición de los requisitos da como resultado un pliego de condiciones, una descripción detallada de cómo se han de cumplir los requisitos del proyecto, así como un plan para la prueba de aceptación, entre otros.

Por último, la primera fase del waterfall model incluye un **análisis de la definición de los requisitos** en el que los problemas complejos se dividen en pequeñas tareas secundarias y se elaboran las correspondientes estrategias de resolución.

Diseño

La fase de diseño sirve para formular una solución específica en base a las exigencias, tareas y estrategias definidas en la fase anterior. En esta fase, los desarrolladores de software se encargan de diseñar la **arquitectura de software**, así como un **plan de diseño detallado del mismo**, centrándose en componentes concretos, como interfaces, entornos de trabajo o bibliotecas. La fase de diseño da como resultado un borrador preliminar con el plan de diseño del software, así como planes de prueba para los diferentes componentes.

Implementación

La arquitectura de software concebida en la fase de diseño se ejecuta en la **fase de implementación**, en la que se incluye la **programación del software**, la **búsqueda de errores** y las **pruebas unitarias**. En la fase de implementación, el proyecto de software se traduce al correspondiente lenguaje de programación. Los diversos componentes se desarrollan por separado, se comprueban a través de las **pruebas unitarias** y se integran poco a poco en el producto final. La fase de implementación da como resultado un producto de software que se comprueba por primera vez como producto final en la siguiente fase (prueba alfa).

Prueba

La fase de prueba incluye la integración del software en el entorno seleccionado. Por norma general, los productos de software se envían en primer lugar a los usuarios finales seleccionados en **versión beta** (pruebas beta). Las **pruebas de aceptación** desarrolladas en la fase de análisis permiten determinar si el software cumple con las exigencias definidas con anterioridad. Aquellos productos de software que superan con éxito las pruebas beta están listos para su lanzamiento.

Servicio

Una vez que la fase de prueba ha concluido con éxito, se autoriza la **aplicación productiva** del software. La última fase del modelo en cascada incluye la **entrega**, el **mantenimiento** y la **mejora del software**.

Ventajas y desventajas del modelo en cascada

Ventajas	Inconvenientes
✔ Una estructura sencilla gracias a unas fases de proyecto claramente diferenciadas.	✗ Por norma general, los proyectos más complejos o de varios niveles no permiten

	su división en fases de proyecto claramente diferenciadas.
✔ Buena documentación del proceso de desarrollo a través de unos hitos bien definidos.	✗ Poco margen para realizar ajustes a lo largo del proyecto debido a un cambio en las exigencias.
✔ Los costes y la carga de trabajo se pueden estimar al comenzar el proyecto.	✗El usuario final no se integra en el proceso de producción hasta que no termina la programación.
✔ Aquellos proyectos que se estructuran en base al modelo en cascada se pueden representar cronológicamente de forma sencilla.	✗En ocasiones, los fallos solo se detectan una vez finalizado el proceso de desarrollo.

MODELO V

El modelo V o modelo en cuatro niveles es un modelo empleado en diversos procesos de desarrollo, por ejemplo, en el desarrollo de software. En los años 90 apareció su primera versión, pero con el tiempo se ha ido perfeccionando y adaptando a los métodos modernos de desarrollo. La idea básica, sin embargo, se remonta a los años 70 y fue concebida como una especie de desarrollo posterior del modelo de cascada.

Además de las fases de desarrollo de un proyecto, el modelo V también define los procedimientos de gestión de la calidad que lo acompañan y describe cómo pueden interactuar estas fases individuales entre sí. Su nombre se debe a su estructura, que se asemeja a la letra V.

LAS FASES DEL MODELO V

En primer lugar, el modelo V define el curso de un proyecto en fases individuales cada vez más detalladas:

- Al principio del proyecto, el modelo prevé un análisis de las especificaciones del sistema planificado (fase de especificaciones).
- El proyecto se completa después con requisitos funcionales y no funcionales para la arquitectura del sistema (fase funcional).

- A esta fase le sigue el diseño del sistema, en el que se planifican los componentes y las interfaces de este (fase de diseño).
- Una vez completadas estas fases, se puede diseñar en detalle la arquitectura del software (codificación).

Es ahora cuando, de acuerdo con estos planes, comienza el desarrollo en sí del software. A continuación, tendrán lugar las fases de **control de la calidad**, también llamadas de verificación o validación, que siempre están relacionadas con cada una de las fases de desarrollo. El método V abarca las siguientes tareas:

- Pruebas de unidad
- Pruebas de integración
- Integración del sistema
- Validación

INTERACCIÓN ENTRE EL DESARROLLO Y LA VERIFICACIÓN

La “V” del nombre del modelo hace referencia a la forma como el modelo **compara las fases de desarrollo con las fases de control de la calidad correspondientes**. El brazo izquierdo de la letra V contiene las tareas de diseño y desarrollo del sistema, y el derecho las medidas de control de calidad de cada fase. En la unión entre los dos brazos, se sitúa la implementación del producto. En los proyectos de software, esto se refiere a la programación del software.

La implementación correcta de la **arquitectura de software** planificada se comprueba mediante **pruebas de unidad**. Aquí se verifica en detalle si los módulos individuales del software cumplen exactamente las funciones requeridas y proporcionan realmente los resultados que se esperan. Para evitar errores, se recomienda realizar estas pruebas en paralelo al desarrollo.

Las **pruebas de integración** examinan el **diseño del sistema**. Aquí se verifica si cada uno de los componentes interactúa con el resto según lo planificado –por ejemplo, si todos los procesos proporcionan los resultados esperados. En este punto, un resultado incorrecto podría indicar un problema de interfaz.

La **prueba del sistema** verifica si se han cumplido los requisitos generales del sistema definidos al diseñar la **arquitectura del sistema.** En general, estas pruebas tienen lugar en un entorno de pruebas que simula las condiciones reales del cliente con la mayor precisión posible.

Al final del proyecto, al **análisis de los requisitos** se contrapone la **validación del producto** terminado. En ella, el cliente comprueba si se cumplen las especificaciones durante el funcionamiento. Como regla general, solo se prueba el rendimiento del

software de forma superficial, es decir, se comprueba lo que el cliente ve durante el uso diario. Esto también se conoce como prueba de validación.

VENTAJAS Y DESVENTAJAS DEL MODELO V

Ventajas	Inconvenientes
✔ Optimización de la comunicación entre las partes involucradas a través de términos y responsabilidades claramente definidos.	✗ El **modelo en cuatro niveles** puede ser demasiado simple para mapear todo el proceso de desarrollo desde el punto de vista de los desarrolladores. Está sobre todo centrado en la **gestión de proyectos**. Además, su estructura relativamente rígida permite una respuesta **poco flexible** a los cambios durante el desarrollo, y, por lo tanto, promueve un curso lineal del proyecto. Sin embargo, si el modelo se entiende y se utiliza correctamente, es posible utilizar el modelo V para el desarrollo ágil.
✔ Minimización de riesgos y mejor planificación a través de roles, estructuras y resultados fijos y predeterminados.	
✔ Mejora de la calidad del producto gracias a medidas de control de la calidad firmemente integradas.	
✔ Ahorro de costes gracias al procesamiento transparente a lo largo de todo el ciclo de vida del producto.	

MODELO INCREMENTAL

El modelo incremental combina elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos. Se basa en la filosofía de construir incrementando las funcionalidades del programa. Este modelo aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software.

Cuando se utiliza un modelo incremental, el primer incremento es a menudo un producto esencial, sólo con los requisitos básicos. Este modelo se centra en la entrega de un producto operativo con cada incremento. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para la evaluación.

VENTAJAS	DESVENTAJAS
✔ Mediante este modelo se genera software operativo de forma rápida y en	✗ Cada fase de una iteración es rígida y no se superponen con otras.

etapas tempranas del ciclo de vida del software.	
✔ Es un modelo más flexible, por lo que se reduce el coste en el cambio de alcance y requisitos.	✗Pueden surgir problemas referidos a la arquitectura del sistema porque no todos los requisitos se han reunido, ya que se supone que todos ellos se han definido al inicio
✔ Es más fácil probar y depurar en una iteración más pequeña	
✔Es más fácil gestionar riesgos.	
✔Cada iteración es un hito gestionado fácilmente	

MODELO RAD

La idea principal es entregar sistemas de alta calidad, en poco tiempo y con un coste bajo de inversión. Para conseguir esto, hay que seguir determinadas pautas que harán que estemos ante una auténtica metodología DRA (las siglas en castellano: Desarrollo Rápido de Aplicaciones), que veremos más adelante.

En la actualidad, las empresas invierten gran parte de sus recursos en desarrollar aplicaciones que les permitan trabajar de forma más eficiente. Con la aparición de los modelos de desarrollo rápido de aplicaciones, podremos crear softwares de forma rápida y barata para satisfacer las necesidades empresariales si n invertir tanto tiempo y dinero.

FASES DENTRO DE UN PROCESO CON METODOLOGÍA RAD

Para implantar el modelo de desarrollo rápido de aplicaciones, hay que seguir una metodología concreta que incluye las siguientes fases, las cuales son cíclicas:

PLANIFICACIÓN DE NECESIDADES:

En esta primera fase, lo que habrá que hacer será sentar las bases de las necesidades del proyecto, tanto de necesidades de la aplicación como de alcance del proyecto y de esta manera comenzar a trabajar en la creación de prototipos.

DISEÑO Y FEEDBACK CON EL USUARIO:

Los usuarios aportarán comentarios que serán determinantes a la hora de diseñar la arquitectura del sistema. Con el feedback del usuario crearemos modelos y prototipos iniciales. Y este paso se podrá repetir tantas veces como se considere necesario según avance el proyecto.

CONSTRUCCIÓN:

Ahora que tenemos el diseño básico, tendremos que realizar el grueso del proyecto: Codificación, pruebas e integración reales de la aplicación. Al igual que en la fase anterior, esta se podrá repetir tantas veces como necesitemos, según haya nuevos componentes o modificaciones en el proyecto.

TRANSICIÓN:

La última fase, también conocida como “cutover”, permitirá al equipo de desarrollo pasar los componentes a un entorno de producción en vivo, para realizar todas las pruebas que sean necesarias.

En la actualidad, existen herramientas que aplican el desarrollo RAD o desarrollo low code, en su forma de trabajar. Una de ellas es Mendix, si quieres saber qué es Mendix no dejes de visitar el post del link.

Con todo esto, creemos que será más fácil comprender por qué usar RAD es una buena elección para tu forma de trabajar.

VENTAJAS	DESVENTAJAS
✔ Avances medibles: Al contar con numerosas iteraciones, componentes y prototipos desplegados cada cierto tiempo, se podrá medir y evaluar de forma sencilla el desarrollo del proyecto y, así, cumplir con los presupuestos.	✗ Requiere sistemas modulares: Cuando aplicamos el método RAD, cada componente del sistema debe ser iterable y constatable por sí mismo, para poder ser modificados o intercambiados por cualquier miembro del equipo.
✔ Productivos más pronto: La metodología DRA permitirá a los desarrolladores adoptar roles multidisciplinares que creen prototipos y códigos de trabajo de forma rápida, lo que supone ser productivos más rápido.	✗ Dificultad dentro de proyectos a gran escala: Cuando estemos ante un proyecto que implique muchas personas y aplicaciones, la flexibilidad puede llegar a ser un problema puesto que perderemos ligeramente el control sobre el diseño y el desarrollo.
✔ Separación de los componentes del sistema: La metodología RAD exige a los diseñadores y desarrolladores a generar componentes funcionales e independientes por sí mismos, y así poder usarlos en en una versión o prototipo iterativo. De esta manera, cada elemento se reparte en compartimentos y se podrá	✗ Exige mucha interactividad del usuario: Conseguir feedback del usuario desde una etapa temprana es muy útil pero, a la vez, puede ser una espada de doble filo ya que tendremos que aceptar todo tipo de críticas constructivas y ser competente a la hora de comunicarse con los usuarios

modificar según evolucionen las necesidades del software y/o usuario.	
✔ Comentarios constantes de los usuarios: Al poder lanzar prototipos e iteraciones ágilmente, obtendremos un feedback muy valioso por parte de los usuarios de forma continuada.	
✔ Integración temprana de sistemas: Los softwares desarrollados con la metodología RAD podrán ser integrados casi desde el comienzo con otros sistemas. A diferencia de los softwares desarrollados en cascada que deben esperar prácticamente al final del desarrollo a ser integrados. Al poder realizar estas integraciones tempranas, podremos identificar los posibles errores que existan en las mismas y buscar la solución.	✗ Necesidad de desarrolladores senior: Aplicar la metodología RAD no es tan fácil como parece, por lo que en el equipo serán necesarios desarrolladores hábiles que sean capaces de aplicar y adaptarse a cualquier necesidad o cambio.
✔ Adaptabilidad: Gracias al desarrollo rápido de aplicaciones, el software es bastante maleable, lo que nos beneficiará para poder realizar cualquier posible adaptación a los prototipos o iteraciones.	

MODELO ÁGIL

El desarrollo de software ágil es un concepto usado en el desarrollo de software para describir las metodologías de desarrollo incrementales (Cohen, Lindvall & Costa, 2003). Es una alternativa en la gestión tradicional de proyectos TI, donde se hace hincapié en el empoderamiento de las personas para colaborar y tomar decisiones en equipo, además potencia la planificación continua, pruebas permanentes y la integración conjunta del código y los despliegues.

Como parte de este modelo, se han agrupado varias metodologías de desarrollo, las cuales se basan en el Manifiesto Ágil. Este es un documento elaborado en febrero de 2001 por líderes y expertos de la industria del software, que está basado en la experiencia de lo que funciona y no funciona en dicho campo de la ingeniería.

Las técnicas ágiles varían en prácticas y énfasis, pero comparten características comunes, incluyendo el desarrollo iterativo y un enfoque en la interacción, la comunicación y la reducción de artefactos intermedios que consumen muchos recursos.

Desarrollar en iteraciones permite al equipo adaptarse rápidamente a las necesidades cambiantes. Trabajando en instalaciones cercanas y centrándose en la comunicación, permite que los equipos puedan tomar decisiones y actuar sobre ellas de inmediato, en lugar de esperar respuestas de otras instancias posteriores.

La reducción de los artefactos intermedios que no agregan valor a la entrega final representa más recursos, que se pueden dedicar al desarrollo del producto en sí y su terminación oportuna.

RECOMENDACIONES DE UNA METODOLOGÍA DE SOFTWARE ÁGIL

Ahora, aplicado al entorno competitivo y real para la ejecución de un proyecto bajo metodologías ágiles, se deben contemplar estas recomendaciones:

1. Es imperativa la participación de los usuarios.
2. El equipo de desarrollo debe tener la facultad para tomar decisiones.
3. Los requisitos evolucionan, pero la escala de tiempo y fechas de entregas son fijas (control del alcance).
4. Capturar los requisitos a un alto nivel, ligero y visual (prototipos).
5. Desarrollar versiones pequeñas, incrementales e itere sobre ellas.
6. Enfocarse en la entrega frecuente de productos.
7. Completar cada funcionalidad antes de pasar a la siguiente.
8. Aplicar la regla 80/20, trabajar funcionalidades principales – principio de Pareto.
9. Las pruebas se integran en todo el ciclo de vida del proyecto – prueba temprano y con frecuencia.
10. Un enfoque de colaboración y cooperación entre todas las partes interesadas es esencial.

COMPENDIO DE METODOLOGÍAS ÁGILES

Actualmente se conocen diferentes metodologías relacionadas con el agilismo, como una respuesta a la creciente necesidad de la industria por entregar productos de calidad en el menor tiempo y costo posible.

Entre ellas, se pueden mencionar:

- EXTREME PROGRAMMING (XP)

Metodología que se centra en el desarrollo, en lugar de aspectos de gestión de proyectos. XP fue diseñada para que las organizaciones tuvieran libertad de adoptar la totalidad o parte de la metodología.

Los proyectos XP comienzan con una fase de planificación de entregables “release plan”, seguido de varias iteraciones, cada una de las cuales termina con pruebas de aceptación

del usuario. Cuando el producto tiene suficientes características para satisfacer a los usuarios, el equipo termina la iteración y libera el software.

Los usuarios escriben «historias de usuario» – user stories – para describir la necesidad que el software debe cumplir. Dichas historias ayudan al equipo a estimar el tiempo y los recursos necesarios para construir el entregable y para definir las pruebas de aceptación. El usuario o su representante hacen parte del equipo de XP, de tal forma que puede agregar detalles sobre los requisitos, mientras el software está siendo construido en sus iteraciones.

Para crear un plan de despliegue, el equipo divide las tareas de desarrollo en iteraciones, cada una de ellas con su propio plan. Al final de una iteración, los usuarios realizan pruebas de aceptación, mapeando contra las historias de usuario. Si encuentran errores, corregir los fallos se convierte en una actividad para la siguiente iteración.

Se caracteriza también por la programación por pares y la integración continua (diaria) de código fuente en una línea base del proyecto.

- SCRUM

Fue propuesta por Ken Schwaber y Jeff Sutherland en 1995. Actualmente es la metodología ágil más usada y extendida en el mundo. Surge de las iniciativas de prototipado rápido, bajo un entorno en que los requisitos se encuentran incompletos en el inicio y son cambiantes durante el desarrollo, A diferencia de XP, la metodología Scrum incluye tanto los procesos de gestión como de desarrollo.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al solicitante del proyecto. Por ello, Scrum está especialmente pensado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en las que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costos se disparan o la calidad es inaceptable; cuando se necesita capacidad de reacción ante la competencia, cuando la rotación de los equipos es alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de productos.

Se compone de: planeación de la iteración, ejecución y cierre de la iteración (demostración y retrospectiva)

- LEAN DEVELOPMENT

Según esta metodología, la perfección se persigue a través de la reducción de las actividades sin valor agregado, pero también a través de perfeccionar el flujo y eliminado la sobrecarga.

La definición de Lean aplicada a software es un reto, porque no hay ningún método o proceso específico asociado. Lean no es comparable con el resto de las metodologías explicadas en este artículo. Podría llamarse Lean a un proceso del ciclo de vida de desarrollo de software o a un proceso de administración de proyectos, sólo si se comprueba que cumple los valores y principios establecidos para dicho concepto. Para implementarlo, debe crear o ajustar su propio proceso de desarrollo de software, comprendiendo los principios y adoptando los valores básicos de Lean.

- KANBAN

La metodología se enfoca en mejorar la visibilidad del flujo de trabajo, limitar el trabajo en curso de acuerdo con la capacidad disponible y medir el tiempo de ciclo de vida de una actividad.

Para ello se divide el trabajo en bloques, cada actividad se clasifica dentro de alguno de los bloques y se ubica en un tablero general, el cual se conoce como tablero Kanban. Para cada bloque se definen los límites de trabajo en progreso que van a ser los umbrales para la ejecución de actividades o acciones en cada etapa definida. Usualmente los bloques se definen como trabajo pendiente, en progreso y completado. Las actividades se incorporan al flujo de trabajo de manera constante sin restricción alguna, salvo cuando se sobrepase el límite o umbral, para lo cual se miden en tiempo real los cuellos de botella y el lead time – tiempo de una actividad para ser completada.

En Kanban, los equipos multi-funcionales son opcionales, y el tablero no necesita estar en manos de un equipo específico. Un tablero está relacionado con un flujo de trabajo, no necesariamente con un equipo o proyecto.

MODELO ITERATIVO

Es un modelo derivado del ciclo de vida en cascada. Este modelo busca reducir el riesgo que surge entre las necesidades del usuario y el producto final por malos entendidos durante la etapa de recogida de requisitos.

Consiste en la iteración de varios ciclos de vida en cascada. Al final de cada iteración se le entrega al cliente una versión mejorada o con mayores funcionalidades del producto. El cliente es quien después de cada iteración evalúa el producto y lo corrige o propone

mejoras. Estas iteraciones se repetirán hasta obtener un producto que satisfaga las necesidades del cliente.

Este modelo se suele utilizar en proyectos en los que los requisitos no están claros por parte del usuario, por lo que se hace necesaria la creación de distintos prototipos para presentarlos y conseguir la conformidad del cliente.

<table>
<tr><th>VENTAJAS</th><th>DESVENTAJAS</th></tr>
<tr><td>✔ Una de las principales ventajas que ofrece este modelo es que no hace falta que los requisitos estén totalmente definidos al inicio del desarrollo, sino que se pueden ir refinando en cada una de las iteraciones.</td><td rowspan="2">✗ La primera de las ventajas que ofrece este modelo, el no ser necesario tener los requisitos definidos desde el principio, puede verse también como un inconveniente ya que pueden surgir problemas relacionados con la arquitectura.</td></tr>
<tr><td>✔ Igual que otros modelos similares tiene las ventajas propias de realizar el desarrollo en pequeños ciclos, lo que permite gestionar mejor los riesgos, gestionar mejor las entregas…</td></tr>
</table>

MODELO DE PROTOTIPO

Un modelo prototipo o modelo de desarrollo evolutivo es utilizado principalmente en el desarrollo de *software* para ofrecer al usuario una visión previa de cómo será el program**a** o sistema. Se le dice de desarrollo evolutivo al modelo de prototipo porque evoluciona hasta convertirse en el producto final.

En un modelo de prototipos las características fundamentales son:

- **Tiempo**. El prototipo se desarrolla en menos tiempo para poder ser probado o testeado.
- **Coste**. La inversión en un modelo de prototipo es ajustada, lo que requiere un uso óptimo de los recursos.
- **Conciso**. El prototipo debe incluir los requisitos y características básicas de la aplicación para poder evaluar su funcionamiento y utilidad.
- **Evolutivo**. El prototipo evoluciona gracias a la interacción con los usuarios.
- **Funcional**. El prototipo es una aplicación que funciona.

El modelo de prototipos se centra en un diseño rápido que representa las características principales del programa que el usuario podrá ver o utilizar. De esta manera pueden

probarlo y dar su opinión sobre distintos aspectos como la usabilidad, la utilidad o el rendimiento, entre otras.

El prototipo se puede modificar cuando sea necesario y todos los resultados obtenidos de las presentaciones y pruebas se deben anotar para utilizar posteriormente como ayuda en el desarrollo del producto final.

ETAPAS PARA LA ELABORACIÓN DEL MODELO DE PROTOTIPO

Un modelo de construcción de prototipos comienza con la definición de un problema y sus efectos, para poder desarrollar el prototipo que lo resuelva. Las etapas para la elaboración del modelo de prototipo son:

1. REQUISITOS DE DESARROLLO

Se realiza un análisis para poder establecer cuáles son los requisitos del programa. Se trata de un diseño básico del prototipo donde se traza de forma inicial los requisitos necesarios para su desarrollo.

2. MODELAJE Y DESARROLLO DEL CÓDIGO

En esta fase se construye el prototipo inicial según los requisitos establecidos. En esta fase de diseño y construcción se debe priorizar el tiempo de desarrollo y hacer un uso óptimo de los recursos para reducir su coste.

3. EVALUACIÓN

Una vez desarrollado el prototipo es necesario comprobar su funcionamiento, evaluando su funcionalidad y verificando que cumple realmente con los requisitos iniciales.

4. MODIFICACIÓN

Tras evaluar el prototipo se deben corregir los errores encontrados y aplicar las mejoras necesarias para que esté listo para ser probado por los usuarios.

5. DOCUMENTACIÓN

Todo el diseño y desarrollo debe ser documentado para disponer de información precisa y clara del proceso. Es muy importante el registro de cada paso o acción del desarrollo del prototipo pues es una guía útil a la hora de afrontar el diseño del producto final.

6. PRUEBAS

Finalmente, el prototipo debe ser probado por los usuarios para poder recibir el *feedback* necesario y así evaluar su utilidad y rendimiento. Gracias a esta retroalimentación

ofrecida por el prototipo se podrá desarrollar un *software* de mayor calidad que resuelva los problemas de los usuarios.

TIPOS DE MODELO DE PROTOTIPOS

Existen diferentes tipos de modelo de prototipos que se utilizan dependiendo del tipo de producto a desarrollar o el objetivo que se persigue en el desarrollo.

Los principales tipos de modelo de prototipos son:

MODELO DE PROTOTIPOS RÁPIDO

En este modelo se da prioridad al desarrollo rápido pues el objetivo es conseguir una rápida evaluación del modelo, por ejemplo, para evaluar si es viable o rentable el desarrollo del producto final.

EVOLUTIONARY PROTOTYPING

El modelo de prototipos reutilizables tiene como objetivo poder utilizarlo posteriormente en el desarrollo del *software* final (en su totalidad o algunas de sus partes).

INCREMENTAL PROTOTYPING

El modelo de prototipos modular o prototipado incremental va añadiendo nuevas funcionalidades, características o elementos, a medida que el diseño del prototipo progresa

MODELO DE PROTOTIPOS HORIZONTAL

En este modelo se abarcan muchos aspectos y funciones del programa, aunque la mayoría de ellas no están operativas. El objetivo es medir el alcance del producto, no su grado de funcionalidad (su uso real).

MODELO DE PROTOTIPOS VERTICAL

Al contrario que el modelo horizontal, en este modelo se persigue medir la operatividad del programa, por lo que se incluyen solo un pequeño número de funciones para evaluar su funcionamiento real.

Los prototipos y modelos de desarrollo de conceptos son una metodología que permite evaluar la utilidad y funcionalidad de un programa antes de abordar su desarrollo, gracias al *feedback* de los usuarios al probarlo.

Para la creación de programas informáticos, *apps* móviles o el desarrollo de proyectos web, utilizar un modelo de prototipos aporta beneficios como reducir el riesgo, controlar el coste de desarrollo, mejorar la experiencia del usuario y conseguir un producto de mayor calidad en menor tiempo.

MODELO ESPIRAL

El modelo de desarrollo en Espiral es una combinación entre el modelo waterfall y un modelo por iteraciones.

El proceso pasa por distintas etapas, desde la de conceptualización, siguiendo el desarrollo, luego una fase de mejoras, para finalizar con el mantenimiento.

Dentro de cada etapa, tendremos una serie de fases que transcurren desde la planificación, pasando por el análisis de riesgos, el desarrollo y finalizando en la evaluación de lo realizado. Se incorpora también una fase de enlace entre etapas, para facilitar la transición entre las mismas.

En definitiva, el equipo de desarrollo en este modelo de desarrollo en espiral comienza con un pequeño conjunto de requisitos y pasa por cada fase de desarrollo para ese conjunto de requisitos. El equipo de desarrollo agrega la funcionalidad para el requerimiento adicional en espirales cada vez mayores, hasta que la aplicación está lista para la fase de producción.

FASES DEL MODELO ESPIRAL

PLANIFICACIÓN

Incluye la estimación del coste, el calendario y los recursos para la iteración.

Implica también la comprensión de los requisitos del sistema para la comunicación continua entre el analista de requerimientos y el cliente.

ANÁLISIS DEL RIESGO

La identificación de los riesgos potenciales se realiza mientras se planifica y finaliza la estrategia de mitigación de riesgos.

INGENIERÍA

Incluye la codificación, pruebas y el despliegue del software.

EVALUACIÓN

Evaluación del software por parte del cliente.

Además, incluye la identificación y el seguimiento de riesgos tales como los retrasos en los plazos y los sobrecostes.

¿CUÁNDO DEBERÍAS USAR EL DESARROLLO EN ESPIRAL?

El uso del método en espiral, como cualquier otra aproximación al desarrollo de software, tiene escenarios en los que se devuelve mejor. Luego, cada caso puede tener sus excepciones, cada organización por su propia estructura puede beneficiar el rendimiento de unas sobre otras, así que esto debéis tomarlo como un escenario generalista, pero que debe ser estudiado en profundidad caso por caso.

Aparentemente los beneficios del uso del modelo en espiral son más destacados en un entorno donde:

- El proyecto es grande.
- Se quiere que las liberaciones de software sean frecuentes.
- Aplica la creación de un prototipo.
- Es primordial un control de riesgos y costos.
- En proyectos catalogados de riesgo medio-alto y alto.
- Los requisitos son poco claros y complejos.
- Hay un alto grado de cambios y estos pueden aparecer en cualquier momento.
- El compromiso de proyecto a largo plazo está comprometido, bien sea por razones económicas u otras.

VENTAJAS	DESVENTAJAS
✔ La funcionalidad adicional o los cambios se pueden hacer en una etapa posterior.	✗ Riesgo de no cumplir con la planificación o el presupuesto.
✔ La estimación del coste se hace fácil, ya que la construcción del prototipo se hace en pequeños fragmentos.	✗ Funciona mejor para proyectos grandes, aunque en estos también requiera de una estricta evaluación de riesgos.
✔ El desarrollo continuo o repetido ayuda en la gestión de riesgos.	✗ Para su buen funcionamiento, el protocolo del modelo en espiral debe ser seguido estrictamente.

✔ El desarrollo es rápido y las características se añaden de forma sistemática	✗ Se genera más documentación al tener fases intermedias.
✔ Siempre hay espacio para atender los comentarios de los clientes.	✗ No es aconsejable para proyectos pequeños, la ratio coste beneficio no es rentable.

SINTESIS DE REQUERIMIENTOS

Para todos los involucrados de manera directa o indirecta en el proyecto el presente proyecto nace a partir de la necesidad de agilizar el llamado “armado de horarios”, dicho método ha sido usado durante periodos de inscripción, los cuales siendo observados de una manera algorítmica y junto a los conocimientos de programación.

El objetivo principal de nuestro proyecto es crear una herramienta que facilite la creación de un horario de materias semestrales/cuatrimestrales para los estudiantes de la universidad, así como una aplicación que nos permita reseñar a los profesores mostrar los horarios que tendrán disponibles en el semestre en curso, con esto esperamos que los estudiantes tengan una herramienta útil, precisa y segura para poder crear diversos horarios que se adecuen a sus necesidades.

ENTORNO TECNOLÓGICO

- Big data.
- Machine learning.
- Sistemas de recomendación.
- Aplicación móvil.
- Aplicaciones nativas.
- Aplicaciones web.
- Aplicaciones híbridas.

BASES DE DATOS

- Azure sql database.
- Oracle database cloud service database
- Enterprise edition.

REQUISITOS DE HARDWARE

ESPECIFICACIONES DE LOS EQUIPOS INDIVIDUALES DE DESARROLLO:

Opción1:

- Modelo: G1107TK701B-BE-W3P
- Memoria RAM: 16 GB
- Tipo de memoria: DDRA SDRAM
- Disco duro: 1TB
- Procesador: Intel Core i7-7700
- Tarjeta gráfica: ZOTAC GeForce GTX 1070 Ti 8GB GDDR5 256-bit

- Tarjeta Madre: MSI B350 PC MATE

Opción 2:

- Modelo: Asus Vivo AiO M241DAK-WA014M
- Procesador: AMD® Ryzen™ 5 3500U APU
- Memoria RAM: 16GB DDR4
- Almacenamiento: 512GB SSD M.2
- Gráfica: AMD® Radeon™ Vega 8 Graphics

SERVIDORES

Es necesario tener servidores capaces de trabajar con una gran cantidad de acciones simultáneas, que además permitan al usuario tener distintas opciones para obtener información acerca de las distintas opciones del programa, así como para descargar los horarios que haya generado. Para esto es necesario tener en cuenta que hay distintos servidores que cumplen diferentes características.

- Servidor email: Funciona como una especie de oficina de correo para almacenar
- Servidoresweb: Un servidor web se ocupa de guardar la información en formato HTML.
- De bases de datos: Son dispositivos diseñados para almacenar grandes cúmulos de información.

SERVIDOR DE OVHCLOUD.

Una opción muy buena viene de la empresa OVHcloud, específicamente hablando del servidor Infra-1. Este servidor es perfecto para el procesamiento de aplicaciones «single core». Su procesador Intel Xeon E-2274G posee la mayor frecuencia de base de nuestras gamas de servidores dedicados. En adición de su alto rendimiento, esta CPU incorpora Intel Software Guard Extensions (SGX), la tecnología de seguridad basada en hardware que protege el tratamiento y el intercambio de datos. Nos ofrece:

- Memoria de 32 Gb DDR4 ECC a 2666 Mhz (expansible).
- 12TB HDD SATA, 1 Gb/s de ancho de banda público y 2Gb/s de ancho de banda privada.
- Direcciones IPv4 e IPv6.
- Backup gratuito de 500 Gb.
- Alojamiento en Canadá.

Y en cuanto a la funcionalidad y tareas que le podemos dar a estos servidores tenemos:

- Cálculos matemáticos complejos.
- Manejo de interfaces web.
- Manejo de datos de sitios web.
- Tato seguro de gran cantidad de información

HERRAMIENTAS APLICACIONES MÓVILES

Appcelerator Titanium.

Está creada por la plataforma Appcelerator y es un software idóneo para desarrollar aplicaciones móviles similares a las nativas de los sistemas operativos móviles. Emplea JavaScript y se encarga de traducir automáticamente la programación al resto de sistemas.

PhoneGap.

Sistema pensado para desarrollar aplicaciones multiplataforma empleando exclusivamente HTML5, CS33 y JavaScript. Es actualmente la herramienta del mercado que más plataformas soporta, además de que permite el acceso a gran parte de los elementos de nuestro *smartphone* como la cámara, los contactos o la base de datos.

jQuery Mobile.

Herramienta basada en un framework que, con el uso de HTML5 optimizado para móviles táctiles, permite la adaptación a los distintos aparatos y tamaños de las pantallas.

REQUISITOS FUNCIONALES Y NO FUNCIONALES DEL SISTEMA

Esta herramienta lo que trata de crear, es una ayuda a los alumnos que necesitan tener los horarios “ideales” tanto para trabajar y estudiar. El Generador de Horarios es un programa que de manera gráfica te muestra las materias y sus respectivos datos que servirán a cada alumno/usuario a elegir sus materias en los mejores horarios.

Este programa necesita las materias establecidas cada semestre por la universidad y los datos básicos del alumno. Así mismo se establecen una serie de requerimientos no funcionales los cuales son necesarios para la elaboración y ejecución de la aplicación elaborada estos requerimientos son necesarios en la creación del software para que de esta manera su desarrollo se haga de una manera adecuada y sea un hecho su terminación.

Requisitos funcionales	Requisitos no funcionales
Dar de alta a un usuario	Los datos de la aplicación solo podrán ser modificados por aquellas personas autorizadas para ello
Visualizar las materias disponibles	La licencia de uso de software donde se aloje y con el que se realice la aplicación debe ser lo menos restrictiva posible
Seleccionar las materias a tu elección	El software deberá tener una estructura clara ordenando el contenido y datos de la aplicación en apartados que abarquen todas las funcionalidades disponibles
Buscar materias disponibles	El software funcionara offline pero necesitara de una conexión a internet para actualizar los datos
Visualizar las materias elegidas	El software deberá proporcionar tiempos de respuesta rápido

	El software debe proporcionar seguridad al usuario

INGENIERÍA DE REQUERIMIENTOS

El proceso de recopilar, analizar y verificar las necesidades del cliente para un sistema de software es llamado Ingeniería de Requerimientos. La meta de la ingeniería de requerimientos es entregar una especificación de requerimientos de software correcta y completa. La ingeniería de requerimientos apunta a mejorar la forma en que comprendemos y definimos sistemas de software complejos. Existen varias definiciones de requerimientos.

Un requerimiento es el proceso mediante el cual se intercambian diferentes puntos de vista para recopilar y modelar lo que el sistema va a realizar. Este proceso utiliza una combinación de métodos, herramientas y actores, cuyo producto es un modelo del cual se genera un documento de requerimientos.

Algunas técnicas para desarrollar/plantear algunos de los requerimientos para el proyecto son:

Entrevistas frente a frente

La técnica más común para reunir requerimientos es sentarte con los clientes y preguntarles que es lo que necesitan.

Entrevistas grupales

Son similares a las entrevistas frente a frente excepto que hay más de una persona siendo entrevistada. Las entrevistas grupales requieren más preparación y más formalidad para obtener la información que quieres de todos los participantes.

Sesiones facilitadas

En una sesión facilitada, reúnes a un grupo más grande por un propósito común. En este caso, intentas reunir un conjunto común de requerimientos del grupo de una manera más rápida que si entrevistaras a cada uno por separado.

Sesiones de desarrollo conjunto de aplicaciones

Las sesiones de desarrollo conjunto de aplicaciones (JAD) son similares a las sesiones facilitadas. Sin embargo, el grupo típicamente se queda en la sesión hasta que se acuerda y documenta un conjunto de requerimientos.

Cuestionarios

Son buenas herramientas para reunir requerimientos de los interesados en locaciones remotas o de los que tendrán una muy pequeña participación en los requerimientos generales. Puede también ser la única manera práctica de reunir requerimientos de docenas, cientos o miles de personas.

Prototipos

Es una manera de construir una versión inicial de la solución – un prototipo. Se lo muestras al cliente, quien te da requerimientos adicionales.

Seguir a las personas

Esto es especialmente útil cuando reúnes información sobre procesos actuales. Quizá requieras ver a las personas desempeñar su trabajo antes de poder entender la idea general. En algunos casos quizá quieras también participar en el proceso real de rebajo para obtener una experiencia cercana de cómo funciona el negocio actualmente.

Se desarrollan sistemas de software para satisfacer una necesidad percibida por un cliente. Pueden formularse las necesidades reales del cliente como requerimientos. Los requerimientos son desarrollados conjuntamente por el cliente, usuario y diseñadores del sistema de software. La ingeniería de requerimientos es un proceso de descubrimiento, refinamiento, modelización, especificación y validación de lo que se desea construir. En este proceso tanto el cliente como el analista juegan un papel muy importante.

Cuando nos encontramos al frente de un proyecto de desarrollo de sistemas es importante dejar claramente definidos los requerimientos del software, en forma consistente y compacta, esta tarea es difícil básicamente porque consiste en la traducción de unas ideas vagas de necesidades de software en un conjunto concreto de funciones y restricciones. Además, el analista debe extraer información dialogando con muchas personas y cada una de ellas se expresará de una forma distinta, tendrá conocimientos informáticos y técnicos distintos, y tendrá unas necesidades y una idea del proyecto muy particulares.

Es justamente este último punto uno de los que se atacará en esta tesis: que el usuario y el desarrollador compartan el mismo lenguaje asegura la comunicación entre ambos sugiere que en particular el uso del lenguaje propio del usuario mejora considerablemente esta comunicación. En el proceso de Ingeniería de Requerimientos la validación de los diferentes productos requiere una fuerte interacción con el usuario, la que se ve facilitada por el vocabulario común usuario-desarrollador. Las diferentes representaciones que se construyen en el proceso de desarrollo de software encuentran en el vocabulario del usuario un marco referencial que permite al desarrollador, obtener un vocabulario de trabajo que es un subconjunto de la terminología del cliente, lo que a su vez facilita el acceso a la documentación por parte de todos los participantes en el

desarrollo. En muchos casos es difícil especificar los requerimientos de un problema, con la mayor calidad posible en una etapa temprana del proceso de desarrollo.

La construcción de prototipos constituye un método alternativo, que da como resultado un modelo ejecutable del software a partir del cual se pueden refinar los requerimientos. Para poder construir el prototipo se necesitan técnicas y herramientas especiales.

El análisis de requerimientos establece el proceso de definición de requerimientos como una serie de tareas o actividades mediante las cuales se busca ganar conocimientos relevantes del problema, que se utilizarán para producir una especificación formal del software necesario para resolverlo. En este proceso se deben conciliar diferentes puntos de vista y utilizar una combinación de métodos, personas y herramientas. El resultado final constituye la documentación de los requerimientos. Éstos deben expresarse de forma clara y estructurada de manera que puedan ser entendidos tanto por expertos como por el usuario, quien deberá participar en la validación.

Como resultado del análisis se desarrolla la especificación de requerimientos del software. La revisión es esencial para asegurar que el realizador del software y el cliente tengan la misma percepción del sistema.

Una vez finalizado nuestro proyecto de desarrollo de sistemas, y contando con un buen análisis de requerimientos, podremos evaluar la calidad del producto terminado.

SINTESÍS DE DISEÑO DEL PROYECTO

Diseño del proyecto:

Diagrama de uso: Para su uso como usuario, teniendo al principio una ventana de Inicio de Sesión la cual simplemente incluye un campo para ingresar el usuario y un campo para escribir la contraseña. Una vez hecho esto, nos encontramos con un panel donde se incluye una lista de materias disponibles para el usuario que inició sesión con la posibilidad de seleccionar una de la lista junto a los datos básicos de la materia (Nombre de profesor, NRC, días y horas de clase).

Dependiendo de la calificación anteriormente obtenida en el semestre, se mostrará un panel donde se podrá encontrar el horario y la selección de materias, aquí podemos añadir la materia que necesitemos de la lista anteriormente mencionada, de acuerdo a el horario necesitado, se muestra para mejor acomodo de materias.

Cuando el horario esté parcialmente finalizado o totalmente finalizado, se podrá mostrar que días, en qué horas y en qué salón están seleccionadas las materias elegidas.

Diagrama de Actividades:

El primer paso dentro del generador de horarios es el inicio de sesión, este paso requiere que el usuario introduzca su usuario y su contraseña para poder loguearse dentro del programa de manera correcta. Este paso de verificación proporciona 2 opciones:

- Cargar datos: Permite iniciar la sesión de un usuario que ya se ha registrado con anterioridad al ingresar el usuario y contraseña indicados.
- Guardar datos: Esta opción permite el registro dentro del programa para los nuevos usuarios. Sus datos quedarán almacenados de forma que la próxima vez que ingresen al programa sea a través de la primera opción.

En cualquiera de los dos casos, se hace una verificación de los datos ingresados, es decir, se verifica si el usuario y la contraseña son correctos.

Posteriormente se carga la interfaz principal que contiene el panel de datos de las materias ya ingresadas con anterioridad. Este panel tiene 3 opciones: iniciar, llenar y mostrar datos. A su vez, este panel está relacionado a otros 2 paneles: Panel de materias y Panel de Horario. Dentro del panel de materias se tiene la opción de activar, mientras que en el panel de horario se puede llenar, vaciar y guardar los horarios realizados.

Al ingresar una materia nueva se solicitan los siguientes datos: nombre, NRC, clave, salón, sección, profesor, calificación y días. Estos datos son tanto los datos que se muestran de cada materia como los que se solicitan para agregar una nueva dentro de un horario. Además, también se tienen las listas de materias, la cual también permite que se busque alguna materia en específico. También se cuenta con una selección que depende del semestre y de la materia en particular que se necesite.

El panel del horario nos permite seleccionar algún horario ya creado con anterioridad, llenar materias y el horario tarde o temprano. En cuanto a los datos de los días, se tiene: día, hora, duración y salón.

Estas opciones se consideran son las necesarias y más fundamentales para una correcta interacción entre los usuarios y el programa.

PATRONES DE DISEÑO

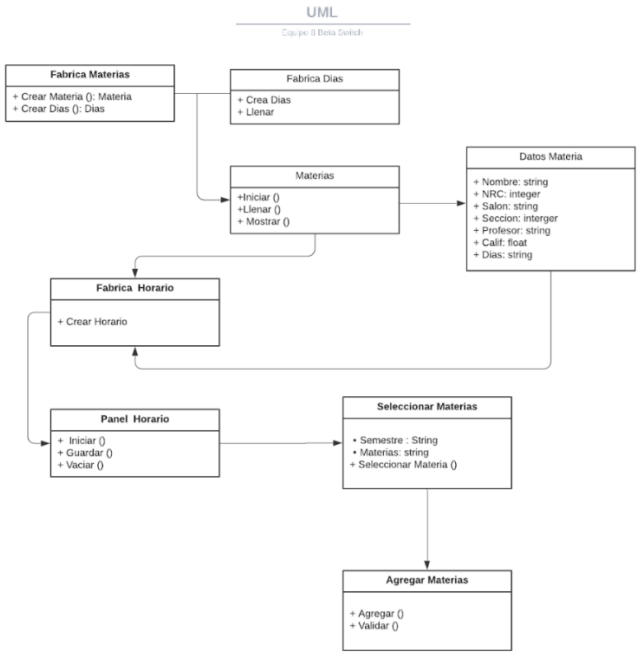
Un patrón de diseño es, de forma resumida, una solución que se puede aplicar a diferentes problemáticas a la hora de diseñar un software. Se consideran “plantillas” que permiten identificar errores y problemas dentro de un sistema y que proporcionan respuestas a estos problemas generales, que de manera normal se hubiera resuelto a través de la prueba y error, implicando así una gran cantidad de tiempo en el análisis de los programas.

El uso de estos patrones se lleva a cabo durante la fase de desarrollo, ya que es en este momento donde es óptimo verificar si existen errores dentro del programa para darles una solución rápida y que no generen nuevos fallos.

Existen 3 tipos de patrones de diseño de software, los cuales son:

- Creacionales: Permiten que el código sea más flexible y reutilizable
- Estructurales: Permiten dar soluciones rápidas.
- De comportamiento: Muestran patrones de comunicación entre objetos y clases.

Dentro de la clase de patrones creacionales se encuentra el patrón denominado “Abstract Factory”, el cuál crea objetos que pueden ser utilizados en diferentes clases o familias. Este patrón es el que se eligió para verificar nuestro desarrollo de software dentro del proyecto.
Diagramas



En este caso, tenemos nuestra fábrica principal llamada “Materias”, a partir de esta tenemos la fábrica de “Días”, donde una crea las materias y otra llena los días de cada materia, ambas creando un día donde se une la materia y el día de la semana para dicha materia.
De estas 2, desciende la variante de “Materia” donde contamos con todos los datos usados y por usar en cada una, tales como: Nombre, NRC, el salón o edificio, sección, profesor y calificación.

Teniendo esto en cuenta, tenemos la variante del panel de horario donde se inicia con una interfaz visual donde se agregan las materias, armar un prototipo de horario y/o vaciar el prototipo por cualquier motivo.

Más a fondo de este panel estará el Seleccionar Materias donde se muestra su Nombre y el Semestre en el que se cursa. Ya en los pasos finales se encuentra el Agregar Materias, donde se valida si se puede cursar la materia dependiendo varios factores (el cupo, si está desbloqueada y se adapta al horario) para finalmente crear tu horario con las materias elegidas y en los días y horas mejor acomodadas.

IMPLEMENTACIÓN Y CALIDAD.

La calidad se mide de distintas formas dependiendo del tipo de producto que se vaya a producir, sin embargo, no solo está presente en el producto final, ya que se deben considerar todos los procesos que son necesarios para el desarrollo. Inicialmente las guías para saber cómo y en qué aspectos medir la calidad son brindadas por las especificaciones y requisitos marcados por el cliente y es importante recordar que la calidad es proporcional a la satisfacción del cliente. Por esto, en el ámbito de software se han analizado distintos modelos y opciones que aseguren la mayor calidad posible en el desarrollo de productos que involucren software.

Al conjunto de las buenas prácticas que intervienen en el ciclo de vida del software se le conoce como modelo de calidad del software. Estas acciones deben estar enfocadas en los procesos de gestión y desarrollo de proyectos, por lo que indican los pasos para saber qué hacer y cómo no hacerlo.

De forma general existen tres diferentes enfoques de calidad que se le pueden dar a estos modelos. Estos enfoques se centran en tres distintos aspectos de alta importancia dentro del desarrollo de proyectos de software, los cuales son:

- Calidad de producto: Son propiedades evaluadas según los desarrolladores y los usuarios tomando como base de análisis la lista de requerimientos y especificaciones de los clientes.
- Calidad de proceso: Son todas aquellas actividades que influyen dentro de la elaboración del producto en cuestión, desde la organización hasta el mantenimiento.
- Calidad en uso: En este caso la prioridad es el análisis de la relación entre el producto y el ambiente de uso para el que fue pensado.

Cabe destacar que los dos primeros enfoques de calidad tienen un valor enfocado en los aspectos técnicos y de desarrollo, mientras que el último está pensado para medir el valor comercial que tendrá el producto cuando ya esté en la etapa de uso. Dentro de la industria los modelos de calidad basados en procesos son los más dominantes y los más requeridos.

Calidad basada en productos:

Algunos de los modelos más usados en cuanto a modelos a nivel producto se tienen:

- McCall.
  Uno de los modelos pioneros en la evaluación de la calidad de software, tiene tres etapas definidas: factores, criterios y métricas. Los once criterios base, son: Exactitud, confiabilidad, eficiencia, integridad, usabilidad, mantenibilidad, testeabilidad, flexibilidad, portabilidad, reusabilidad e interoperabilidad (Khosravi, 2004).

- Gilb.
  Modelo de calidad que orienta la evaluación de software a partir de los atributos: Capacidad de trabajo, adaptabilidad, disponibilidad y utilizabilidad, los cuales se dividen en subatributos, de tal manera que sirva de apoyo a la gestión de proyectos, y proporcione una guía para solucionar problemas y detectar riesgos (Khosravi, 2004).

- ISO 25000.
  También llamadas como SQuaRE, cuyo propósito es guiar el desarrollo con los requisitos y la evaluación de atributos de calidad, principalmente: la adecuación funcional, eficiencia de desempeño, compatibilidad, capacidad de uso, fiabilidad, seguridad, mantenibilidad y portabilidad (Alfonso, 2012).

Calidad basada en procesos:

En este caso se busca analizar las actividades de algún proceso específico en donde la calidad del producto se vea afectada en gran parte o donde más se podría perder. Por esto, dicho proceso en específico se modela para poder llevar el proceso de análisis de una manera más precisa. En esta etapa se pueden realizar preguntas como:

- ¿dónde y cuándo se puede hallar un tipo de defecto?
- ¿cómo prevenir los defectos?
- ¿Hay alguna actividad distinta que proporcione mayor calidad?

Algunos de los modelos basados en procesos son los modelos de madurez, tales como:

- CMM (Capability Madurity Model) y CMMI (CMM Integrated).
  El CMM - CMMI es un modelo de calidad del software que clasifica las empresas en niveles de madurez. Estos niveles sirven para conocer la madurez de los procesos que se realizan para producir software. El CMM está compuesto de 316 prácticas claves agrupadas en 18 áreas y distribuidas en una jerarquía de cinco niveles, a través de los cuales una organización progresivamente alcanza mayor calidad, productividad y menores costos en el desarrollo de software. Los niveles progresan desde el 1, que representa el estado caótico, hasta el nivel 5, que representa el estado de optimización continua

- ISO 15504 SPICE (Software Process Improvement and Capability determination).
  La norma SPICE establece requisitos para una evaluación de procesos y los modelos de evaluación pretendiendo que estos requisitos puedan ser aplicados en cualquier modelo de evaluación en una organización. En general, los requisitos para la evaluación de procesos comprenden:

  - Evaluación de procesos
  - Mejora de procesos
  - Evaluación de capacidad y/o madurez de los procesos.

- NYCE NMX-I-050/02 (Moprosoft y Evalprosoft) Norma Mexicana.
  Esta Norma Mexicana tiene por objeto definir el modelo de procesos para la industria de software. MoProSoft está dirigido a las organizaciones dedicadas al desarrollo y mantenimiento de software. Es aplicable tanto para las organizaciones que tienen procesos establecidos, así como para las que no cuenten con ellos.

Una vez el proceso de desarrollo de software está completo y a pesar de haber seguido alguna de las metodologías antes mencionadas, se debe hacer una verificación y validación de los productos para asegurar que el software sea acorde a las especificaciones y necesidades de los clientes.

La fase de verificación y validación es un proceso de ciclo de vida completo que inicia con las revisiones de los requerimientos y continúa con las revisiones del diseño y las inspecciones del código hasta la prueba del producto en el contexto para el que fue desarrollado. Es importante llevar a cabo la validación de los requerimientos del sistema de forma inicial. Es fácil cometer errores y omisiones durante la fase de análisis de requerimientos del sistema y, en tales casos, el software final no cumplirá las expectativas de los clientes. Sin embargo, en la realidad, la validación de los requerimientos no puede descubrir todos los problemas que presenta la aplicación.

Dentro del proceso de Verificación y validación se utilizan dos técnicas de comprobación y análisis de sistemas:

1. Las inspecciones del software: analizan y comprueban las representaciones del sistema como el documento de requerimientos, los diagramas de diseño y el código fuente del programa. Se aplica a todas las etapas del proceso de desarrollo. Las inspecciones se complementan con algún tipo de análisis automático del texto fuente o de los documentos asociados. Las inspecciones del software y los análisis automatizados son técnicas de verificación y validación estáticas puesto que no requieren que el sistema se ejecute.

2. Las pruebas del software. consiste en contrastar las respuestas de una implementación del software a series de datos de prueba y examinar las respuestas del software y su comportamiento operacional, para comprobar que se desempeñe conforme a lo requerido. Llevar a cabo las pruebas es una técnica dinámica de la verificación y validación ya que requiere disponer de un prototipo ejecutable del sistema.

El proceso de inspección siempre se realiza utilizando una lista de los errores comunes de los programadores. Esta se somete a discusión por el personal con experiencia y se actualiza frecuentemente según se vaya teniendo experiencia. La lista de errores debe ser función del lenguaje de programación que se utilice. Por ejemplo, un compilador Ada comprueba que las funciones tienen el número correcto de argumentos, mientras que C no. Muchos de los fallos en C tienen relación con los punteros, estos no se pueden producir en Java.

De manera general, otras pruebas que se pueden realizar de forma más específica son:

- Proceso de depuración: Se localizan y corrigen los errores que se descubrieron durante la fase de verificación y validación general. Utiliza una metodología de prueba y error.
- Inspección del software: Son revisiones sistemáticas de los documentos generados como de los códigos fuentes con el único propósito de detectar fallos.

- Lista de fallos: Aquí se registran todos los fallos descubiertos durante los procesos anteriores.
- Inspección del código fuente: Se utilizan analizadores estáticos para detectar fallos que los compiladores ignoran.
- Pruebas de software: Se dividen en pruebas de unidades, de integración, de sistema y de validación.

INTEGRACIÓN, VERIFICACIÓN Y VALIDACIÓN DEL SOFTWARE

Integración de Sándwich:

Son una combinación de enfoques de arriba hacia abajo y de abajo hacia arriba.

Elegimos esta prueba de integración ya que es una combinación donde las pruebas y el objetivo se juntan para tener el modelo ideal de nuestro programa.

Verificación:
Para hacer la verificación se responde la pregunta de: ¿Se está construyendo adecuadamente?

Esto puede ser subjetivo, debido a la manera de programar de cada uno. Lo que se busca es tener el programa listo de una forma “ideal” a nuestra perspectiva como programadores. Una vez funcionando, buscamos la optimización del código y reducirlo para facilitar las revisiones futuras, esto para minimizar riesgos durante su uso ya implementado.

Nuestro programa trata de no ser robusto, ya que no es necesario, incluso que sea posible una aplicación web y facilitar su implementación, uso y mejoras.

Prueba de Stress:

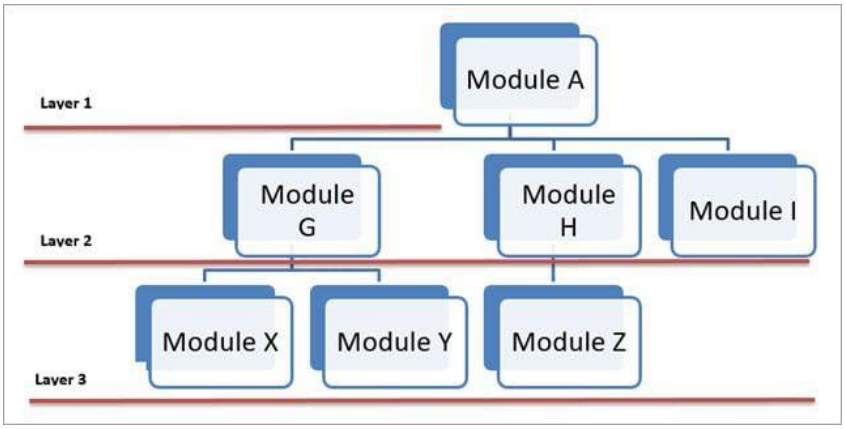
Elegimos este tipo de prueba ya que nos permite saber el alcance de nuestro programa en condiciones extremas o anormales. Cosas que, aunque son posibles, son el peor escenario donde el código se pone a prueba con todos los recursos disponibles. Es ahí donde vemos que tanta carga soporta.

Prueba de Desempeño:

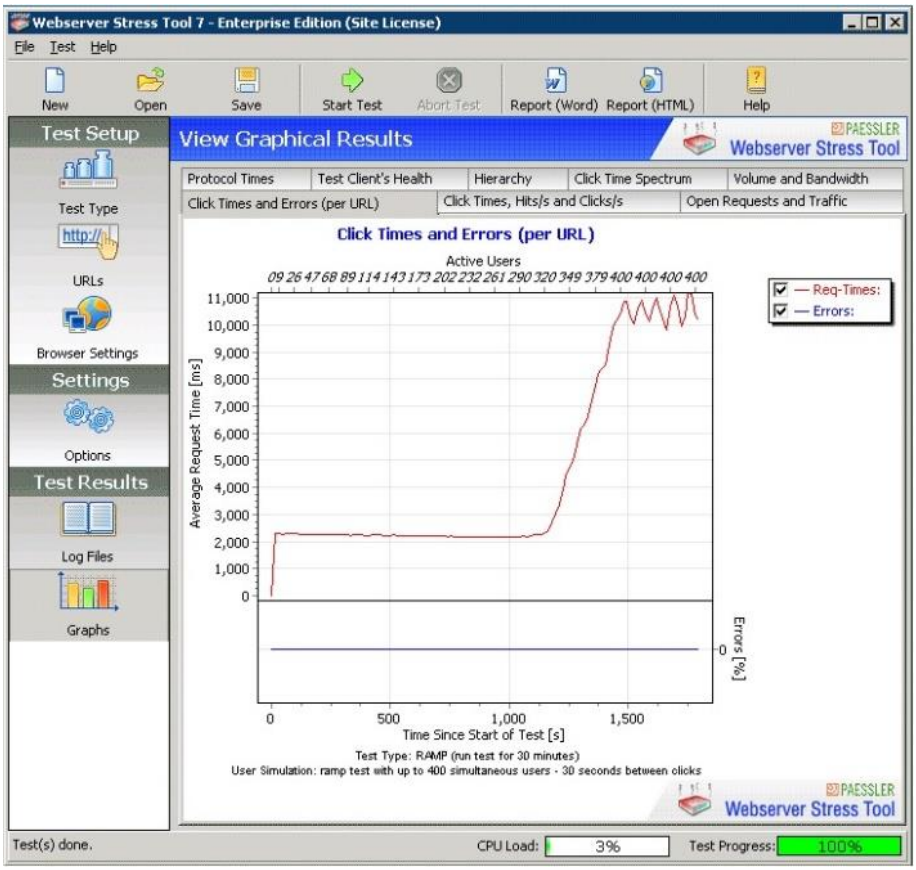
Otra prueba fundamental para identificar el comportamiento de cada llamado a cada proceso es la de desempeño, con esta tendremos un monitoreamiento de nuestro programa en ejecución, teniendo en cuenta en que se ocupa el tiempo de ejecución.

Esta prueba corre en un entorno controlado, nos muestra donde puede haber procesos lentos e ineficientes y regresa un reporte de consumo de tiempo.

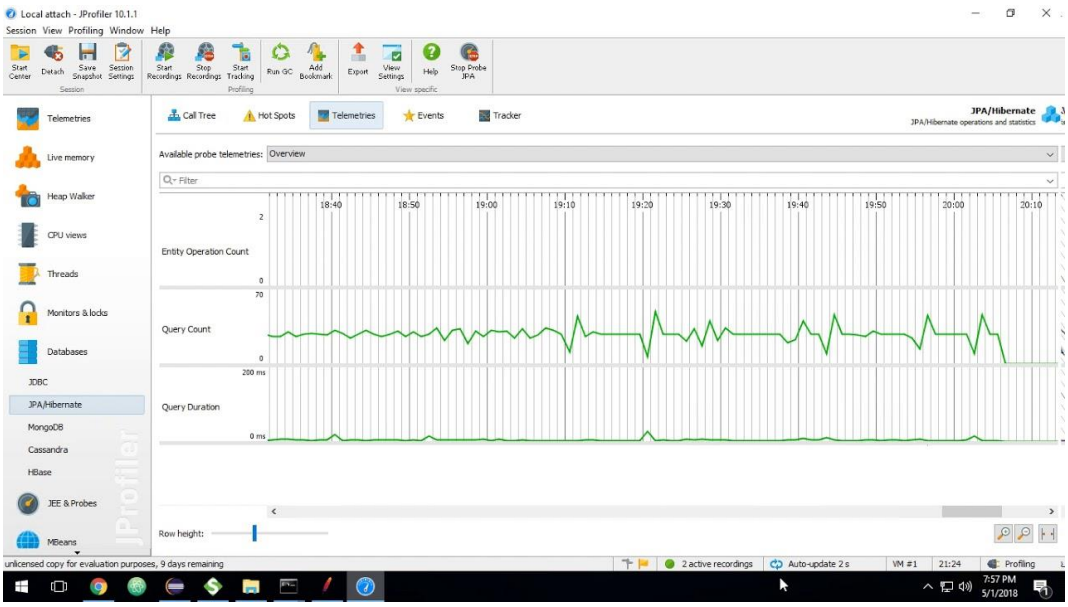
Ejemplo de prueba de integración de sándwich:



Ejemplo de prueba de Stress:



Ejemplo de prueba de desempeño:



Validación:

Para hacer la validación se responde la pregunta de: ¿Se está construyendo el producto adecuado?

Creemos que es algo muy útil para los estudiantes que puede tener un futuro prometedor. El usuario principal requiere un horario y una manera de obtenerlo más sencilla, ahí donde entra nuestro programa ya que, con buena aceptación, mejoras y diferentes versiones puede ayudar no solo a estudiantes si no a cualquiera que necesite organizar un horario de manera más sencilla.

MANTENIMIENTO

El coste del mantenimiento de un producto software a lo largo de su vida útil es superior al doble de los costes de su desarrollo.

Fechas	% Mantenimiento
Años 70	35% - 40%
1980 – 1984	55%
1985 – 1989	75%
Años 90	80% - 90%

Evolución de los costes del mantenimiento

Por norma general, el porcentaje de recursos necesarios en el mantenimiento se incrementa a medida que se genera más software. A esta situación se le conoce como Barrera de Mantenimiento. Las causas a las que se debe este incremento de trabajo de mantenimiento son:

1) Gran cantidad de software antiguo (más de 10 años); aun siendo construidos con las mejores técnicas de diseño y codificación del momento (rara vez), su creación se produjo con restricciones de tamaño y espacio de almacenamiento y con herramientas desfasadas tecnológicamente.
2) Los programas sufren migraciones continuas de plataformas o SSOO.
3) El software ha experimentado modificaciones, correcciones, mejoras y adaptaciones a nuevas necesidades de los usuarios. Además, estos cambios se realizaron sin técnicas de reingeniería o ingeniería inversa, dando como resultado sistemas que funcionan con baja calidad (mal diseño de estructuras de datos, mala codificación, lógica defectuosa y escasa documentación).

Como consecuencia de estos grandes costes, es que el coste relativo de reparar un error aumenta considerablemente en las últimas fases del ciclo de vida del software.

Las razones por las que es menos costoso reparar defectos en las primeras fases del ciclo de vida software son:

- Es más sencillo cambiar la documentación que modificar el código.
- Un cambio en las fases posteriores puede repercutir en cambiar toda la documentación de las fases anteriores.
- Es más sencillo detectar un error en la fase en la que se ha introducido que detectarlo y repararlo en fases posteriores.
- Un defecto se puede ocultar en la inexistencia o falta de actualización de los documentos de especificación o diseño.

Existen otra serie de costes intangibles del mantenimiento del software, que son:

- Oportunidades de desarrollo que se han de posponer o que se pierden debido a los recursos dedicados a las tareas de mantenimiento.
- Insatisfacción del cliente cuando no se le satisface en un tiempo debido una solicitud de reparación o modificación.
- Los cambios en el software durante el mantenimiento también introducen errores ocultos.
- Perjuicios en otros proyectos de desarrollo cuando la plantilla tiene que dejarlos o posponerlos debido a una solicitud de mantenimiento.

En conclusión, la productividad en LDC (líneas de código) por persona y mes durante el proceso de mantenimiento puede llegar a ser 40 veces inferior con respecto al proceso de desarrollo.

TIPOS DE MANTENIMIENTO

Existen 4 tipos de mantenimiento:

- Correctivo.
- Adaptativo.
- Perfectivo.
- Preventivo.

Mantenimiento correctivo

Tiene por objetivo localizar y eliminar los posibles defectos de los programas. Un defecto en un sistema es una característica del sistema con el potencial de provocar un fallo. Un fallo se produce cuando el comportamiento de un sistema difiere con respecto al comportamiento definido en la especificación.

- Los fallos en un sistema software pueden ser:
- Procesamiento (salidas incorrectas de un programa).
- Rendimiento (tiempo de respuesta demasiado alto).
- Programación (inconsistencias en el diseño).
- Documentación (inconsistencias entre la funcionalidad de un programa y el manual de usuario). Mantenimiento adaptativo:

Consiste en la modificación de un programa debido a cambios en el entorno (hardware o software) en el que se ejecuta. Desde cambios en el sistema operativo, pasando por cambios en la arquitectura física del sistema informático, hasta en el entorno de desarrollo del software. Este tipo de mantenimiento puede ser desde un pequeño retoque hasta una reescritura de todo el código.

Los cambios en el entorno de desarrollo del software pueden ser:

- En el entorno de los datos (p.e. cambiar sistema de ficheros por BD relacional).
- En el entorno de los procesos (p.e. migración a plataforma con procesos distribuidos).

Este mantenimiento es cada vez más frecuente debido a la tendencia actual de actualización de hardware y SSOO cada poco tiempo.

Mantenimiento perfectivo:

Conjunto de actividades para mejorar o añadir nuevas funcionalidades requeridas por el usuario.

Se divide en dos:

- Mantenimiento de Ampliación: incorporación de nuevas funcionalidades.
- Mantenimiento de Eficiencia: mejora de la eficiencia de ejecución.

Mantenimiento preventivo:

Modificación del software para mejorar las propiedades de dicho software (calidad y mantenibilidad) sin alterar sus especificaciones funcionales. Incluir sentencias que comprueben la validez de los datos de entrada, reestructuración de los programas para aumentar su legibilidad o incluir nuevos comentarios. Este tipo de mantenimiento utiliza las técnicas de ingeniería inversa y reingeniería. El mantenimiento para la reutilización especializado en mejorar la reusabilidad del software se incluye en este tipo.

ACTIVIDADES DE MANTENIMIENTO:

Las actividades de mantenimiento se agrupan en tres categorías funcionales:

Comprensión del software y de los cambios a realizar (Comprender): es necesario el conocimiento a fondo de la funcionalidad, objetivos, estructura interna y requisitos del software. Alrededor del 50% de tiempo de mantenimiento se dedica a esta actividad, a consecuencia de lo cual, las herramientas CASE incorporan utilidades que automatizan este tipo de tareas aumentando de manera notable la productividad.

Modificación del software (Corregir): crear y modificar las estructuras de datos, la lógica de procesos, las interfaces y la documentación. Los programadores deben evitar los efectos laterales provocados por sus cambios. Esta actividad representa ¼ del tiempo total de mantenimiento.

Realización de pruebas (Comprobar): realizar pruebas selectivas que nos aseguren la corrección del software.

CATEGORÍA	ACTIVIDAD	% TIEMPO
Comprensión del software y de los cambios a realizar	Estudiar las peticiones	18%
	Estudiar la documentación	6%
	Estudiar el código	23%
Modificación del software	Modificar el código	19%
	Actualizar la documentación	6%
Realización de pruebas	Diseñar y realizar pruebas	28%

Importancia de las actividades de mantenimiento

CONCLUSIONES

La ingeniería de software es una disciplina muy compleja que involucra demasiados procesos que a su vez contienen muchos factores y elementos que, en su totalidad,

conforman los proyectos de software. Por esto mismo, antes de desarrollar el proyecto específico es necesario entender las partes, metodologías, tipos y conceptos que se necesitan para desarrollar una herramienta que cumpla con todos los estándares de calidad y con los requisitos del cliente. Como bien hemos abarcado en el documento debemos de llevar un proceso abarcando desde el objetivo o problema que se quiere resolver hasta la gestión de esta, por lo que realmente es una serie de cuestiones que implica el desarrollo de un sistema, para ello es muy importante estar enfocados en lo que se está realizando.

Uno de los factores más relevantes a la hora de desarrollar un proyecto de esta índole es la administración del tiempo, actividades y recursos humanos. Esta administración comienza desde las primeras etapas del proyecto en las que se define qué tipo de software se utilizará para posteriormente comenzar con el manejo de los miembros del equipo, es decir, con la asignación de roles considerando el ciclo de vida del software. Además, gracias a la administración de estos recursos la gestión y administración del proyecto se hace más sencilla y precisa. Sin embargo, a pesar de contar con todos estos puntos de análisis, el desarrollo del proyecto sería mucho más complejo si no se hubieran analizado los diferentes modelos de desarrollo de software existentes, ya que estos son la guía que seguir para construir de la manera más eficiente nuestro proyecto.

Una vez estudiado el concepto de ingeniería de requerimientos, se puedo observar la importancia que tiene el conocimiento dentro de esta área tan amplia y vital sin dejar de mencionar que el resultado satisfactorio depende de una intensa comunicación entre clientes y analistas de requerimientos. La ingeniería se encarga de establecer y mantener un acuerdo en que el sistema debe hacer, además proporciona al equipo de desarrollo un entendimiento de los requisitos, hasta definir los límites del sistema.

Es muy importante recalcar que el poder formular una especificación de requerimientos completa y consistente es un paso muy importante para evitar cometer errores en la definición de los requerimientos, ya que los mismos pueden resultar muy difíciles de corregir una vez desarrollado el sistema. De ahí la vital importancia que tiene la ingeniería de requerimientos en generar una adecuada especificación que contemple claramente y sin ambigüedades los requerimientos del sistema a desarrollar, con el fin primordial de evitar que los proyectos fracasen debido a una mala elaboración de la definición y especificación de requerimientos.

Esto significa que una descripción completa de los requerimientos garantiza el desarrollo de un buen producto final. Sin embargo, la obtención y especificación de los requerimientos son dos situaciones problemáticas que todo ingeniero de software enfrenta. Por un lado, se busca resolver las diferencias de comunicación, culturales y técnicas que enfrentan los usuarios y los desarrolladores, en el afán de lograr obtener las necesidades y restricciones que definirán el sistema; y, por el otro, la búsqueda de técnicas, métodos, metodologías y herramientas que permitan obtener una

especificación de los requerimientos entendible tanto para los clientes y usuarios como para los ingenieros involucrados en el diseño.

ANEXO 1 REQUERIMIENTOS DEL PROYECTO

Autores:

BARBOSA CARRILO, ANTONIO A

ESPINOSA VELAZQUEZ, LUIS A

MATEOS ROMERO, PEDRO

RUBIO MARTINEZ, JEDUS D

TALAVERA SANDOVAL, IVÁN

INTRODUCCIÓN

Para todos los involucrados de manera directa o indirecta en el proyecto (esto es: creadores, usuarios), el presente proyecto nace a partir de la necesidad de agilizar el llamado “armado de horarios”, dicho método ha sido usado durante periodos de inscripción, los cuales siendo observados de una manera algorítmica y junto a los conocimientos de programación, surge este proyecto. Como ya se mencionó, el proyecto ayuda a los usuarios (estudiantes), de manera que con los datos que se manejan durante los periodos de inscripción (los cuales son proporcionados por la universidad), son ingresados para ser de cierta forma procesados, para finalmente el programa obtenga, lo que nosotros los estudiantes conocemos como un “horario armado”, para evitar retrasos y contratiempos el día de la inscripción correspondiente. Cabe mencionar que como se menciona en la parte de “OBJETIVO” en el presente documento, el programa busca encontrar un profesor que se adecúe a las necesidades de cada estudiante, lo que de cierta manera reduciría el deserto de estudiantes en sus respectivas materias, lo que se conoce como “dar de baja la materia” y sus respetivas secuelas.

OBJETIVO

El objetivo principal de nuestro proyecto es crear una herramienta que facilite la creación de un horario de materias semestrales/cuatrimestrales para los estudiantes de la universidad así como una aplicación que nos permita reseñar a los profesores mostrar los horarios que tendrán disponibles en el semestre en curso así como las materias que han impartido a lo largo del tiempo con esto esperamos que los estudiantes tengan una herramienta útil, precisa y segura para poder crear diversos horarios que se adecuen a

sus necesidades así como encontrar un profesor que se adecue a las necesidades de los estudiantes.

ENTORNO TECNOLÓGICO

BIG DATA

Cuando hablamos de Big Data nos referimos a conjuntos de datos o combinaciones de conjuntos de datos cuyo tamaño (volumen), complejidad (variabilidad) y velocidad de crecimiento (velocidad) dificultan su captura, gestión, procesamiento o análisis mediante tecnologías y herramientas convencionales, tales como bases de datos relacionales y estadísticas convencionales o paquetes de visualización, dentro del tiempo necesario para que sean útiles.

Esta tecnología va a estar implementada en nuestro proyecto, ya que necesitaremos tomar datos constantemente de todas las féminas que utilicen la pulsera en tiempo real y a cada momento del día.

MACHINE LEARNING

Es una rama de la inteligencia artificial (AI) cuyo objetivo es incorporar en las máquinas algoritmos que aprendan automáticamente y por medio de unas características básicas la máquina pueda por sí mismo hacer una clasificación o tomar una decisión respecto a una entrada de datos.

SISTEMAS DE RECOMENDACIÓN

Sistema de filtrado de información que busca predecir la calificación, clasificación o preferencia que un usuario le daría a un elemento para emitir una recomendación sobre su utilización.

APLICACIÓN MÓVIL

Una móvil, también llamada app móvil**,** es un tipo de aplicación diseñada para ejecutarse en un dispositivo móvil, que puede ser un teléfono inteligente o una tableta. Incluso si las aplicaciones suelen ser pequeñas unidades de software con funciones limitadas, se las arreglan para proporcionar a los usuarios servicios y experiencias de calidad.

A diferencia de las aplicaciones diseñadas para computadoras de escritorio, las aplicaciones móviles se alejan de los sistemas de software integrados. En cambio, cada aplicación móvil proporciona una funcionalidad aislada y limitada. Por ejemplo, puede ser un juego**,** una calculadora o un navegador web móvil**.**

Debido a los recursos de hardware limitados de los primeros dispositivos móviles, las aplicaciones móviles evitaban la multifuncionalidad. Sin embargo, incluso si los

dispositivos que se utilizan hoy en día son mucho más sofisticados, las aplicaciones móviles siguen siendo funcionales. Así es como los propietarios de aplicaciones móviles permiten a los consumidores seleccionar exactamente las funciones que deben tener sus dispositivos.

APLICACIONES NATIVAS

Estas aplicaciones están diseñadas para un único sistema operativo móvil. Por eso se denominan nativos: son nativos de una plataforma o dispositivo en particular La mayoría de las aplicaciones móviles actuales están diseñadas para sistemas como Android o iOS. En pocas palabras, no puedes instalar ni usar una aplicación de Android en iPhone y viceversa.

El principal beneficio de las aplicaciones nativas es su alto rendimiento y excelente Experiencia de Usuario (Ux). Después de todo, los desarrolladores que los crean utilizan la Interfaz de Usuario (Ui) del dispositivo nativo. El acceso a una amplia gama de API también ayuda a acelerar el trabajo de desarrollo y ampliar los límites del uso de la aplicación. Las aplicaciones nativas solo se pueden descargar de las tiendas de aplicaciones e instalarlas directamente en los dispositivos. Es por eso que primero deben pasar por un estricto proceso de publicación.

El inconveniente más importante de las aplicaciones nativas es su costo. Para crear, respaldar y mantener una aplicación para Android y iOS, básicamente necesitas dos equipos de desarrollo. Como puedes imaginarte, esto puede hacer que tu proyecto tenga más gastos.

APLICACIONES WEB

Las aplicaciones web son aplicaciones de software que se comportan de manera similar a las aplicaciones móviles nativas y funcionan en dispositivos móviles. Sin embargo, existen diferencias significativas entre las aplicaciones nativas y las aplicaciones web. Para empezar, las aplicaciones web utilizan navegadores para ejecutarse y, por lo general, están escritas en CSS, HTML5 o JavaScript.

Dichas aplicaciones redirigen al usuario a la URL y luego les ofrecen la opción de instalar la aplicación. Simplemente crean un marcador en su página. Por eso requieren una memoria mínima del dispositivo.

Dado que todas las bases de datos personales se guardarán en el servidor, los usuarios solo pueden usar la aplicación si tienen una conexión a Internet. Este es el principal inconveniente de las aplicaciones web: siempre requieren una buena conexión a Internet. De lo contrario, corre el riesgo de ofrecer una Experiencia de Usuario (Ux) insatisfactoria.

Además, los desarrolladores no tienen tantas API que funcionen, a excepción de las funciones más populares, como la geolocalización. El rendimiento también estará vinculado al trabajo del navegador y la conexión de red.

APLICACIONES HÍBRIDAS

Estas aplicaciones se crean utilizando tecnologías web como JavaScript, CSS y HTML 5. Las aplicaciones híbridas funcionan básicamente como aplicaciones web disfrazadas de un contenedor nativo.

Las aplicaciones híbridas son fáciles y rápidas de desarrollar, lo cual es un claro beneficio. También obtiene una única base de código para todas las plataformas. Esto reduce el costo de mantenimiento y agiliza el proceso de actualización.

Los desarrolladores también pueden aprovechar muchas API para funciones como giroscopio o geolocalización.

Por otro lado, las aplicaciones híbridas pueden carecer de velocidad y rendimiento. Además, es posible que experimente algunos problemas de diseño, ya que es posible que la aplicación no tenga el mismo aspecto en dos o más plataformas.

DESCRIPCIÓN DE HARDWARE

Gracias a que nuestro proyecto no está al nivel de un trabajo empresarial real, los requerimientos de hardware para desarrollar nuestro programa no son excesivamente complejos ni de la mayor calidad posible. Sin embargo, el hardware necesario para la mejor experiencia de uso del programa sí debe ser de alta calidad, ya que de este depende gran parte de la experiencia del usuario, la cual obviamente buscamos que sea la mejor posible. En este caso particular, se nos es posible desarrollar los programas necesarios con los equipos que se poseen actualmente, sin embargo, pensando como una empresa que está interesada en establecer un grupo profesional de desarrollo de software, así como de aplicaciones de uso continuo para una gran cantidad de personas, es necesario pensar en posibles opciones de hardware que nos permitan trabajar de la forma más eficaz posible.

En cuanto al hardware para llevar a cabo la programación de todo lo requerido es necesario tener equipos eficaces, rápidos y que pasen el estándar de procesamiento de datos. La opción más eficiente es utilizar “Workstation” ya que están hechas para realizar múltiples tareas de forma simultánea y ha desarrollado un excelente software que ayuda a cumplirlas con una exactitud impresionante. Sus características permiten mantener y elevar la productividad ya que están hechas para trabajar bajo cualquier circunstancia a cualquier hora, sumado a su gran poder de procesamiento, su escalabilidad y, muy importante, la capacidad para manejarla desde otros sitios (acceso remoto).

ESPECIFICACIONES DE LOS ESUIPOS INDICIDUALES:

Modelo: G1107TK701B-BE-W3P

Memoria RAM: 16 GB

Tipo de memoria: DDRA SDRAM

Disco duro: 1TB

Procesador: Intel Core i7-7700

Tarjeta gráfica: ZOTAC GeForce GTX 1070 Ti 8GB GDDR5 256-bit

Tarjeta Madre: MSI B350 PC MATE

El costo total de estos equipos rondaría los 30,000 pesos, por lo que se deberá considerar la cantidad de equipos que se requerirán y se deberán conseguir y ensamblar por piezas, ya que no existen proveedores de esta clase de equipo preensamblado. En caso de querer conseguir equipos ya armados, se tiene esta otra opción:

Modelo: Asus Vivo AiO M241DAK-WA014M

Procesador: AMD® Ryzen™ 5 3500U APU

Memoria RAM: 16GB DDR4

Almacenamiento: 512GB SSD M.2

Gráfica: AMD® Radeon™ Vega 8 Graphics

En estos equipos no se tienen gráficas en el presupuesto, por lo que se debe considerar si es un requerimiento indispensable o si los gráficos integrados son los suficientes, además de que la capacidad del disco duro es de la mitad a comparación de la primera opción. Sin embargo, su precio es de aproximadamente 16,800 pesos, mucho más bajo que el presupuesto anterior.

SERVIDORES

Es necesario tener servidores capaces de trabajar con una gran cantidad de acciones simultáneas, que además permitan al usuario tener distintas opciones para obtener información acerca de las distintas opciones del programa, así como para descargar los horarios que haya generado. Para esto es necesario tener en cuenta que hay distintos servidores que cumplen diferentes características.

Servidor email: Funciona como una especie de oficina de correo para almacenar, recibir, enviar y permitir múltiples operaciones que tienen que ver con el correo personal de los clientes. Permiten contactar a los clientes/usuarios de manera eficaz y rápida. Además, específicamente se debe usar el servidor POP3, los cuales retienen los emails recibidos hasta que el usuario los abre, momento en que son enviados al dispositivo (computadora, teléfono, Tablet).

Servidores web: Un servidor web se ocupa de guardar la información en formato HTML de los sitios, donde se incluye texto, imágenes, videos y todo tipo de datos. Mediante un explorador web, los usuarios puedan visualizar todo esto en sus pantallas.

De bases de datos: Son dispositivos diseñados para almacenar grandes cúmulos de información y poder gestionar los datos uno por uno. También son capaces de analizar, manipular y alojar los datos de acuerdo a los requerimientos del usuario.

Así, la manera más eficaz y barata de cumplir con estos requerimientos es contratar un servidor de alguna empresa externa que pueda cumplir con estas características primordiales, ya que así se evitan costos de adquisición, mantenimiento, infraestructura y la gestión de técnica vendrá dada por parte del proveedor, además de que el costo de un servidor individual es bastante alto, y dependiendo de su capacidad va de los 13,000 pesos hasta los 30,000.

SERVIDOR DE OVHcloud.

Una opción muy buena viene de la empresa OVHcloud, específicamente hablando del servidor Infra-1. Este servidor es perfecto para el procesamiento de aplicaciones «single core». Su procesador Intel Xeon E-2274G posee la mayor frecuencia de base de nuestras gamas de servidores dedicados. En adición de su alto rendimiento, esta CPU incorpora Intel Software Guard Extensions (SGX), la tecnología de seguridad basada en hardware que protege el tratamiento y el intercambio de datos.

Además, se puede especificar la cantidad de memoria que deseas tener en tu servidor (teniendo un costo extra si es que deseas una ampliación) aunque de estándar tiene una memoria de 32 Gb DDR4 ECC a 2666 Mhz, así como 12TB HDD SATA, 1 Gb/s de ancho de banda público y 2Gb/s de ancho de banda ilimitado para la banda privada. Posee las Direcciones IPv4 e IPv6ya que Todos los servidores se entregan con una dirección IPv4 pública y un rango de direcciones IPv6. De forma opcional, también puede contratar direcciones IPv4 adicionales para su servidor dedicado (hasta 256 direcciones por máquina). Además, cada servidor dedicado cuenta con un espacio de “backup” gratuito de 500 GB, totalmente independiente del servidor, en el que podrá almacenar sus datos y archivos de configuración. Y si lo necesita puede aumentar su capacidad.

Con todas estas especificaciones tendría un costo de 105,69 USD, lo que vendría siendo aproximadamente 2200 MXN por mes.
Otro punto importante es que está alojado en Canadá, por lo que no habrá demasiado problema en los tiempos de espera y carga de la página y de los procesos que ahí se desarrollen.
Sin duda alguna es la mejor opción, ya que estos servidores son útiles para realizar tareas como:

- Cálculos matemáticos complejos.
- Manejo de interfaces web.
- Manejo de datos de sitios web.
- Tato seguro de gran cantidad de información

DESCRIPCIÓN DE SOFTWARE

BASES DE DATOS

AZURE SQL DATABASE.

Es un servicio de base de datos relacional, inteligente y escalable creado para la nube. Optimice el rendimiento y la durabilidad con características dotadas de inteligencia artificial y automatizadas que siempre están actualizadas. Con las opciones de proceso sin servidor y almacenamiento del nivel Hiperescala que modifican automáticamente la escala de los recursos a petición, puede centrarse en la compilación de nuevas aplicaciones sin tener que preocuparse por el tamaño del almacenamiento ni la administración de los recursos.

Costo $1597.68 al mes.

ORACLE DATABASE CLOUD SERVICE DATABASE

Los productos de bases de datos de Oracle ofrecen a los clientes versiones rentables y de alto rendimiento de Oracle Database, el sistema de gestión de bases de datos multimodelo y convergente líder del mundo, así como bases de datos en memoria, NoSQL y MySQL. Permite a los clientes simplificar los entornos de bases de datos relacionales y reducir las cargas de trabajo de gestión.

Diferentes versiones

Standard Edition.

2 OCPU habilitadas. Licencia de base de datos de 2 OCPU.

Hasta 6 OCPU adicionales (adquiridas por separado).

Instancias de bases de datos con 768 GB de RAM, SSD NVMe de 51,2 TB brutos, aprox. 16 TB

ENTERPRISE EDITION.

2 OCPU habilitadas. Licencia de base de datos de 2 OCPU.

Hasta 50 OCPU adicionales (adquiridas por separado).

Instancias de bases de datos con 768 GB de RAM, SSD NVMe de 51,2 TB brutos

HERRAMIENTAS APLICACIONES MÓVILES

Appcelerator Titanium:

Está creada por la plataforma Appcelerator y es un software idóneo para desarrollar aplicaciones móviles similares a las nativas de los sistemas operativos móviles. Emplea JavaScript y se encarga de traducir automáticamente la programación al resto de sistemas. Además, es muy sencilla y no es necesario ser un experto en programación para dominarla, ya que su interfaz muy intuitiva.

Cuenta con servicios en la nube y posibilita desarrollar apps interconectadas con el software y el hardware, permitiendo el uso del micro, la cámara o el GPS. Está disponible para iOs, Android y Blackberry, y las aplicaciones desarrolladas con Appcelerator Titanium permiten los avances tecnológicos más innovadores en el mundo de las aplicaciones móviles como la geolocalización o la realidad aumentada.

La parte negativa de esta herramienta es su complejidad a la hora de maquetar, ya que no cuenta con un HTML inicial donde añadir los controles. Además, hoy en día parte de sus tutoriales y documentos están desactualizados.

Alrededor de 200 millones de usuarios ejecutan aplicaciones desarrolladas con Appcelerator Titanium, albergando cerca de 575 mil desarrolladores. Empresas de la talla de Avis, Zipcar o Adidas emplean esta herramienta para sus aplicaciones móviles.

PhoneGap

Sistema pensado para desarrollar aplicaciones multiplataforma empleando exclusivamente HTML5, CS33 y JavaScript. Es actualmente la herramienta del mercado que más plataformas soporta, además de que permite el acceso a gran parte de los elementos de nuestro *smartphone* como la cámara, los contactos o la base de datos.

Permite preestablecer la navegación a través del buscador que decidamos como Chrome o Firefox.

PhoneGap sí que requiere conocimientos avanzados de desarrollo, concretamente en JavaScript y HTML. Las aplicaciones que se desarrollan con este sistema no alcanzan el rendimiento de las aplicaciones nativas. Para cada plataforma utiliza un sistema distinto, Xcode para Mac y Eclipse para Android.

jQuery Mobile:

Herramienta basada en un framework que, con el uso de HTML5 optimizado para móviles táctiles, permite la adaptación a los distintos aparatos y tamaños de las pantallas.

De uso sencillo para personas acostumbradas a trabajar con HTML, contiene muy buena documentación que facilita aún más su uso.

Las apps que se crean con esta herramienta quedan lejos de poder compararse con aplicaciones nativas, el resultado se aproxima más a una web adaptada a smartphones. Además de que el manejo del CSS es, cuanto menos, complejo.

REQUISITOS GENERALES DEL SISTEMA

Esta herramienta lo que trata de crear, es una ayuda a los alumnos que necesitan tener los horarios “ideales” tanto para trabajar y estudiar, como para aquellos que quieren los mejores profesores en los mejores horarios. Para esto, se necesita tener en cuenta los datos básicos de la materia: Nombre del profesor, NRC, días y horas, así como las propias calificaciones dadas por los mismos alumnos de cursos anteriores, también datos básicos del alumno como: Nombre, Matrícula y calificaciones. Todo esto con el fin de mostrar las materias disponibles establecidas por la universidad. Una vez seleccionado el profesor con su respectiva materia mostrarlo a manera de horario como normalmente sería impreso y así lograr el mejor horario para ingresarlo al sistema de inscripción.

REQUISITOS FUNCIONALES DEL SISTEMA

Descripción: El Generador de Horarios es un programa que de manera gráfica te muestra las materias y sus respectivos datos que servirán a cada alumno/usuario a elegir sus materias en los mejores horarios. Este programa necesita las materias establecidas cada semestre por la universidad y los datos básicos del alumno, tales como nombre, matrícula y calificaciones.

¿Qué hace?

- Dar de alta a un usuario
- Visualizar las materias disponibles
- Seleccionar las materias a tu elección
- Buscar materias disponibles
- Visualizar las materias elegidas

¿Qué no hace?

- Inscribirte automáticamente en autoservicios
- Ver tu historial de horarios antiguos

Para dar de alta a un usuario: Se entra al programa, se elige la opción de inscribirse y se ingresa la matrícula y una contraseña.

Para buscar materias disponibles: Se entra al programa, se inicia sesión, seleccionas la opción de “Buscar materia” y se ingresa por NRC o por profesor.

Para seleccionar materias: Se entra al programa, se inicia sesión, buscas la materia a elegir y está la opción de dar click para seleccionar la materia para agregarla en tu horario.

REQUISITOS NO FUNCIONALES DEL SISTEMA

Se establecen una serie de requerimientos no funcionales los cuales son necesarios para la elaboración y ejecución de la aplicación elaborada estos requerimientos son necesarios en la creación del software para que de esta manera su desarrollo se haga de una manera adecuada y sea un hecho su terminación.

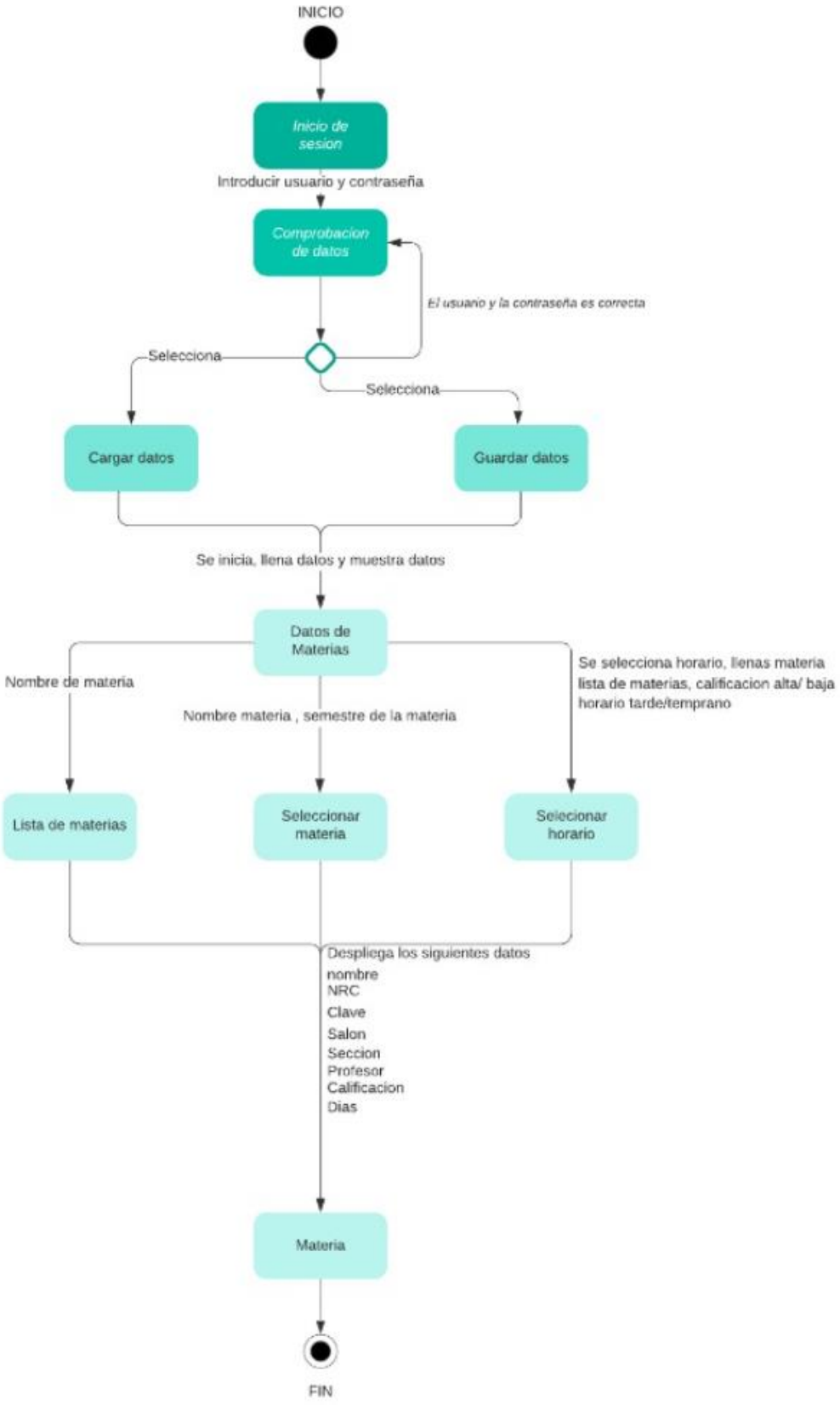
Para el desarrollo del software se tomaron los siguientes requerimientos no funcionales:

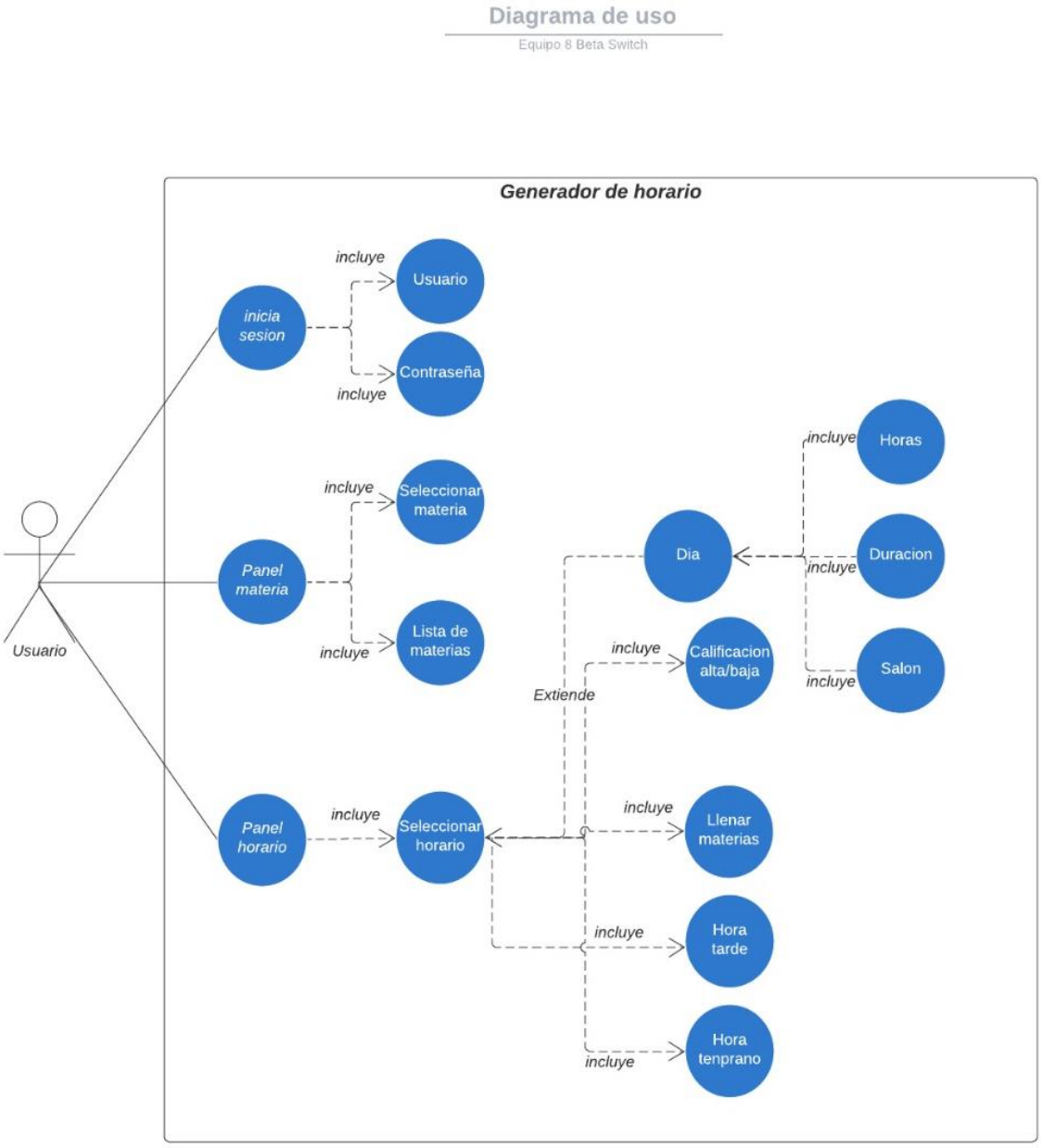
- Los datos de la aplicación solo podrán ser modificados por aquellas personas autorizadas para ello
- La licencia de uso de software donde se aloje y con el que se realice la aplicación debe ser lo menos restrictiva posible
- El software deberá tener una estructura clara ordenando el contenido y datos de la aplicación en apartados que abarquen todas las funcionalidades disponibles
- El software funcionara offline pero necesitara de una conexión a internet para actualizar los datos
- El software deberá proporcionar tiempos de respuesta rápido
- El software debe proporcionar seguridad al usuario

El software está pensado para ejecutarse en un sistema operativo Windows 10 a sea en una laptop o una computadora de escritorio exclusivamente debido al tiempo de desarrollo, contemplando, aun así, la posibilidad de desarrollar una aplicación móvil.

ANEXO 2 DISEÑO DE PROYECTO

DIAGRAMAS:





ANEXO 3: PATRONES DE DISEÑO

Código de los patrones

Fábrica Abstracta:

```
from __future__ import annotations
from abc import ABC, abstractmethod
```

```
class AbstractFactory(ABC):
    @abstractmethod
    def create_product_a(self) -> AbstractProductA:
        pass

    @abstractmethod
    def create_product_b(self) -> AbstractProductB:
        pass


class ConcreteFactory1(AbstractFactory):

    def create_product_a(self) -> AbstractProductA:
        return ConcreteProductA1()

    def create_product_b(self) -> AbstractProductB:
        return ConcreteProductB1()

class ConcreteFactory2(AbstractFactory):

    def create_product_a(self) -> AbstractProductA:
        return ConcreteProductA2()

    def create_product_b(self) -> AbstractProductB:
        return ConcreteProductB2()


class AbstractProductA(ABC):
    @abstractmethod
    def useful_function_a(self) -> str:
        pass

class ConcreteProductA1(AbstractProductA):
    def useful_function_a(self) -> str:
        return "The result of the product A1."


class ConcreteProductA2(AbstractProductA):
    def useful_function_a(self) -> str:
```

```
        return "The result of the product A2."


class AbstractProductB(ABC):

    @abstractmethod
    def useful_function_b(self) -> None:

        pass

    @abstractmethod
    def another_useful_function_b(self, collaborator: AbstractProductA) -> None:
        pass

class ConcreteProductB1(AbstractProductB):
    def useful_function_b(self) -> str:
        return "The result of the product B1."

    def another_useful_function_b(self, collaborator: AbstractProductA) -> str:
        result = collaborator.useful_function_a()
        return f"The result of the B1 collaborating with the ({result})"


class ConcreteProductB2(AbstractProductB):
    def useful_function_b(self) -> str:
        return "The result of the product B2."

    def another_useful_function_b(self, collaborator: AbstractProductA):

        result = collaborator.useful_function_a()
        return f"The result of the B2 collaborating with the ({result})"


def client_code(factory: AbstractFactory) -> None:

    product_a = factory.create_product_a()
    product_b = factory.create_product_b()

    print(f"{product_b.useful_function_b()}")
    print(f"{product_b.another_useful_function_b(product_a)}", end="")
```

```
if __name__ == "__main__":
    print("Client: Testing client code with the first factory type:")
    client_code(ConcreteFactory1())

    print("\n")

    print("Client: Testing the same client code with the second factory type:")
    client_code(ConcreteFactory2())
```

Fábrica de métodos:

```
from __future__ import annotations
from abc import ABC, abstractmethod


class Creator(ABC):
    @abstractmethod
    def factory_method(self):
        pass

    def some_operation(self) -> str:
        product = self.factory_method()
        result = f"Creator: The same creator's code has just worked with
{product.operation()}"

        return result
class ConcreteCreator1(Creator):
    def factory_method(self) -> Product:
        return ConcreteProduct1()

class ConcreteCreator2(Creator):
    def factory_method(self) -> Product:
        return ConcreteProduct2()

class Product(ABC):
    @abstractmethod
    def operation(self) -> str:
        pass
```

```
class ConcreteProduct1(Product):
    def operation(self) -> str:
        return "{Result of the ConcreteProduct1}"


class ConcreteProduct2(Product):
    def operation(self) -> str:
        return "{Result of the ConcreteProduct2}"


def client_code(creator: Creator) -> None:
    print(f"Client: I'm not aware of the creator's class, but it still works.\n"
          f"{creator.some_operation()}", end="")
if __name__ == "__main__":
    print("App: Launched with the ConcreteCreator1.")
    client_code(ConcreteCreator1())
    print("\n")

    print("App: Launched with the ConcreteCreator2.")
    client_code(ConcreteCreator2())
```

Decoradora:

```
class Component():
    def operation(self) -> str:
        pass

class ConcreteComponent(Component):
    def operation(self) -> str:
        return "ConcreteComponent"

class Decorator(Component):
    _component: Component = None

    def __init__(self, component: Component) -> None:
        self._component = component
    @property
    def component(self) -> str:
        return self._component

    def operation(self) -> str:
        return self._component.operation()
```

```
class ConcreteDecoratorA(Decorator):
    def operation(self) -> str:
        return f"ConcreteDecoratorA({self.component.operation()})"


class ConcreteDecoratorB(Decorator):
    def operation(self) -> str:
        return f"ConcreteDecoratorB({self.component.operation()})"


def client_code(component: Component) -> None:
    print(f"RESULT: {component.operation()}", end="")

if __name__ == "__main__":
    simple = ConcreteComponent()
    print("Client: I've got a simple component:")
    client_code(simple)
    print("\n")
    decorator1 = ConcreteDecoratorA(simple)
    decorator2 = ConcreteDecoratorB(decorator1)
    print("Client: Now I've got a decorated component:")
    client_code(decorator2)
```

ANEXO 4: REPOSITORIO

https://github.com/langelespinosa/Generador-de-Horario.git

BIBLIOGRAFÍA

- Chavez, A. [Alan Chavez]. (2020, 3 de Enero). 2.Procesos de software[video]. https://youtu.be/cCD43d-Ndfd0
- "Software de sistema". Autor: Equipo editorial, Etecé. De: Argentina. Para: Concepto.de. Disponible en: https://concepto.de/software-de-sistema/. Última edición: 5 de agosto de 2021. Consultado: 26 de agosto de 2021 - Fuente: https://concepto.de/software-de-sistema/
- AlbertoZnaper. (2012, 12 diciembre). Mitos del Software [Vídeo]. YouTube. https://www.youtube.com/watch?time_continue=13&v=VuE6_96wi98&feature=emb_logo
- Ch1 Introduction Software Engineering book, 10th edition. (2014, 12 diciembre). [Diapositivas]. Slideshare. https://www.slideshare.net/software-engineering-book/ch1-introduction-42645973
- Ingeniería de software: Qué es, Objetivos y Funciones. (2021, 18 agosto). UNIR México. https://mexico.unir.net/ingenieria/noticias/ingenieria-de-software-que-es-objetivos/
- IEEE Standard Glossary of Software Engineering Terminology, IEEE std 610.12-1990, 1990.
- Quintanilla, D. (2019, 31 agosto). La historia de la Ingenieria de software - Ouracademy. Our Academy. https://our-academy.org/posts/software-engineering-history/
- Ciclo de vida del software: todo lo que necesitas saber. (2020, 28 noviembre). Intelequia. https://intelequia.com/blog/post/2083/ciclo-de-vida-del-software-todo-lo-que-necesitas-saber
- Jaramillo, O. (s. f.). *El código de ética y práctica profesional de ingeniería del software*. Https://Www.Slideshare.Net. Recuperado 31 de agosto de 2021, de https://www.slideshare.net/OmarJaramillo13/el-cdigo-de-tica-y-prctica-profesional-de-ingeniera-del-software
- HostingPlus Mexico. (2021, 6 julio). *Modelo de prototipos: ¿qué es y cuáles son sus etapas? | Blog | Hosting Plus Mexico*. Hosting Plus. https://www.hostingplus.mx/blog/modelo-de-prototipos-que-es-y-cuales-son-sus-etapas/
- *El modelo en cascada: desarrollo secuencial de software*. (2021, 16 septiembre). IONOS Digitalguide. https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/
- *Modelo Incremental*. (s. f.). blogspot. Recuperado 19 de septiembre de 2021, de http://isw-udistrital.blogspot.com/2012/09/ingenieria-de-software-i.html
- *El modelo iterativo como evolución del modelo en cascada*. (2020, 1 diciembre). ASPgems. https://aspgems.com/metodologia-de-desarrollo-de-software-ii-modelo-de-diseno-iterativo/

- *¿Qué es el desarrollo en Espiral?* (2020, 8 abril). Deloitte Spain. https://www2.deloitte.com/es/es/pages/technology/articles/que-es-el-desarrollo-en-espiral.html
- *¿Qué es el modelo V?* (2021, 16 septiembre). IONOS Digitalguide. https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/modelo-v/
- *¿Qué es un modelo de proceso de software? Top 7 modelos explicados*. (2021, 8 enero). ICHI.PRO. https://ichi.pro/es/que-es-un-modelo-de-proceso-de-software-top-7-modelos-explicados-108031221097232
- *Qué es y para qué sirve la metodología RAD*. (s. f.). Incentro. Recuperado 19 de septiembre de 2021, de https://www.incentro.com/es-es/blog/stories/metodologia-rad-desarrollo-rapido-aplicaciones/
- Stark, K. (2021, 6 mayo). *¿Qué es desarrollo de software ágil?* Evaluando Software. https://www.evaluandosoftware.com/desarrollo-software-agil/