Estudiantes. Las actividades para el día de hoy consisten en hacer un resumen sobre los siguientes temas, siguiendo la bibliografía indicada y las preguntas orientadoras (para hacer un resumen significativo), y realizando una actividad introductoria a la temática de "Análisis y especificación de requerimientos de software" a fin de que podamos profundizar dichos temas cuando nos encontremos nuevamente:

- **Bibliografía: Pressman. Ingeniería de Software: Un enfoque practico. (Capitulo 4)**
    - **Temas: Principios fundamentales: Principios que guían el proceso. Principios que guían la practica. Principios que guían toda actividad estructural: Principios de Comunicación. Principios de planeación. Principios de Modelado.**
    - **Pregunta orientadora: ¿Qué conceptos y principios guían la práctica de la ingeniería de software?**
- **Bibliografía: Apuntes de catedra del espacio. Capitulo 2.**
    - **Temas: Análisis y especificación de requerimientos. Ingeniería de Requerimientos. Origen de los requerimientos. Definición de requerimiento. Actividades del proceso de obtención de requerimientos.**
- **Bibliografía: Ian Sommerville. Ingeniería de Software.(Parte II: Capitulo 6)**
    - **Temas: Requerimientos del Software: Requerimientos Funcionales, Requerimientos No Funcionales, Requerimientos de Dominio, Requerimientos del usuario, Requerimientos del sistema. Especificación de la Interfaz. El documento de requerimientos del sistema. Procesos de la ingeniería de requerimientos: Estudio de viabilidad, Obtención y análisis de requerimiento, validación de requerimientos, gestión de requerimiento.**

Resumen
Bibliografía de Presman

- Capítulo 4:

Principios fundamentales

La práctica de la ingeniería del software es guiada por principios fundamentales que hacen eficaz, la aplicación de software significativo y métodos eficaces de ingeniería de software

- **Principios que guian el proceso:** Se aplica a la estructura a todo el proceso de software. EStos son:

Principio 1: Ser ágil: Si el modelo es ágil o prescriptivo, de cualquier forma se aplican principios básicos del desarrollo ágil y se mantiene un enfoque técnico y sencillo con resultados concisos, siempre tratando de tomar decisiones locales en lo posible.

Principio 2: En cada etapa, concentrarse en la calidad: Antes de finalizar un proceso, tarea o producto siempre se debe considerar la calidad.
Principio 3: Estar listo para adaptar: El enfoque no es fijo sino que se adapta según los factores.}
Principio 4: Formar un equipo eficaz: El equipo formado debe elegirse según la conveniencia, el respeto y la confianza.
Principio 5: Establecer mecanismos para la comunicación y la coordinación: Los proyectos exitosos conllevan una coordinación y comunicación eficaz sin dejar información de lado.
Principio 6: Administrar el cambio: Se debe establecer una forma óptima de administración de cambios, siendo evaluados, probados e implementados.
Principio 7: Evaluar el riesgo: Se deben tener planes que en caso de que salga mal.
Principio 8: Crear productos del trabajo que agreguen valor para otros: Todo producto como funciones que pasan por etapas hasta su aplicación deben ser desarrolladas de forma completa sin omitir nada para ayudar a la comprensión desde la primera etapa hasta la última etapa.

- **Principios que guían la práctica:** Permiten entregar a tiempo un software de alta calidad , siendo vitales para la práctica para la ingeniería de software. Estos son:

Principio 1: Divide y vencerás: Un gran problema es más sencillo si se divide en partes como entidades , y que cada entidad tenga una función correspondiente , que sea independiente de las demás.
Principio 2: Entender el uso de la abstracción: La abstracción es una forma de simplificar un elemento o información compleja. Y dependiendo el caso varía el nivel de abstracción.
Principio 3: Buscar la coherencia: Al crear un modelo de requerimientos, diseño, o código siempre se debe tener coherencia entre sus partes.
Principio 4: Concentrarse en transferencia de información: En transferencia de información ya sea de una notebook a un sistema externo, de una celu a un sitio web, pueden omitirse algún detalle y por ende llevar la pérdida de la información por hay que prestar mucha atención.
Principio 5: Construir software que tengan modularidad eficaz: La modulación eficaz es dividir al software en componentes que estén ligeramente relacionados y que sus funciones sean prácticamente independientes.
Principio 6: Buscar patrones: El hecho de tener en cuenta estos patrones permite dar soluciones en caso de errores en ciertos patrones , lo que permite que el conjunto de patrones acumulados lleve a crear una estructura ideal y reutilizable.
Principio 7: Cuando es posible representar el problema y su solución desde varias perspectivas diferentes: Al ver un problema desde diferentes ángulos se tienen un visión más amplia de las soluciones.
Principio 8: Tener en cuenta que alguien haga mantenimiento al software: Al tener una práctica sólida de ingeniería, aplicada al nuestro software permite que cuando alguien más deba modificar el código pueda entenderlo.

Principios que guían toda actividad estructural

Principios generales en el éxito de cada actividad estructural genérica, que son similares a lo anteriormente mencionados pero refinados.

- **Principios de comunicación:** Se plantea la comunicación entre el cliente y el project manager y cómo debe hacerse correctamente, aunque también se aplica a otras comunicaciones.

Principio 1: Escuchar: Hay que concentrarse en lo que dice el cliente sin interrupciones, excepto para preguntar una duda que no aclaró el cliente.
Principio 2: Antes de comunicarse, prepararse: Dedicarse a investigar sobre el ámbito donde se aplicará el software y preparar un agenda.
Principio 3: Alguien debe facilitar la actividad: El líder (facilitador) mantienen la conversación en una dirección siendo un mediador en conflictos y siguiendo otros principios.
Principio 4: Es mejor la comunicación cara a cara: Cara a cara puede usarse algún objeto como un gráfico como centro de atención.
Principio 5: Tomar notas y documentar las decisiones: Para no omitir información hay que documentar lo dicho por el cliente.
Principio 6: Perseguir la colaboración: Es importante la comunicación y colaboración entre los integrantes del proyecto para generar confianza.
Principio 7: Permanecer centrado; hacer módulos con la discusión: Mientras más integrantes en la conversación más son propensos a que se distraigan por ello el facilitador debe dar el módulo de conversación para no cambiar de tema y terminar una y después otra.
Principio 8: Si algo no está claro, hacer un dibujo: En caso que la comunicación verbal llegue a su límites se podrá hacer uso de un diagrama o gráfico.
Principio 9: a) Una vez que se acuerde algo, avanzar. b) Sino es posible ponerse de acuerdo en algo, avanzar c) Si una característica no está claro o no puede aclarar en el momento avanzar: Ante la tardanza de tomar una decisión, avanzar solo en casos necesarios, no como en el principio 2 que se annalisa cuidadosamente hasta al final.
Principio 10: La negociación no es un concurso o juego. Funciona mejor cuando las dos partes ganan: Ante negociación de características, fechas, etc, lo ideal es siempre que salgan ganado las dos partes.

- Principios de planeación: esto nos permite el armar un mapa medianamente prediciendo el rumbo del proyecto, esto aplicando prácticas administrativas y técnicas. Entonces ante un planeación rigurosa, que consume mucho tiempo, y una planeación escasa, que lleva al fracaso, siempre se siguen los siguientes principios:

Principio 1: Entender el alcance del proyecto: El mapa debe establecer el alcance o destino.
Principio 2: Involucrar en la actividad de planeación a los participantes del software: Los usuarios y clientes del software son los que definen prioridades y restricciones, por eso, el equipo técnico tiene que negociar con ellos temas como fechas de entrega, funcionalidades más urgentes o limitaciones de recursos
Principio 3: Reconocer que la planeación es iterativa: Ningún plan se mantiene igual desde el principio hasta el final, cuando el desarrollo empieza, las cosas cambian, por eso el plan tiene que adaptarse, y como se trabaja por incrementos, cada vez que se entrega algo se vuelve a planear teniendo en cuenta la experiencia anterior y lo que digan los usuarios
Principio 4: Estimar con base en lo que se sabe: Las estimaciones se hacen con la información real que se tiene, si lo que se conoce es poco o impreciso, la estimación también lo será, por eso mientras más claro se tenga lo que hay que hacer, mejor se puede estimar el tiempo, el costo y el esfuerzo
Principio 5: Al definir el plan, tomar en cuenta los riesgos: Cuando se identifican riesgos que pueden tener mucho impacto y es probable que ocurran, hay que prepararse para eso, se hacen planes de contingencia y se ajustan los cronogramas y tareas para estar cubiertos si algo sale mal
Principio 6: Ser realista: En todo proyecto hay errores, malentendidos, cansancio y cambios de último momento, nadie trabaja perfecto todo el tiempo, por eso al planear se tienen que considerar estos factores, para que el plan no colapse ante la primera complicación
Principio 7: Ajustar la granularidad cuando se defina el plan: La granularidad es el nivel de detalle del plan, para las tareas a corto plazo se necesita mucho detalle, mientras que para las tareas lejanas basta con definirlas de forma general, ya que muchas cosas pueden cambiar, a medida que pasa el tiempo se va afinando el plan con más precisión
Principio 8: Definir cómo se trata de asegurar la calidad: El plan debe indicar claramente qué acciones se van a tomar para asegurar que el software sea de calidad, como revisiones técnicas, pruebas, programación por parejas, o cualquier técnica que ayude a mantener el control sobre el producto
Principio 9: Describir cómo se busca manejar el cambio: El cambio es algo inevitable en el desarrollo, por eso el plan tiene que incluir cómo se van a recibir, analizar y aprobar esos cambios, si el cliente puede pedir cambios en cualquier momento, cómo se decide si se implementan o no, y cómo se calcula su costo e impacto en el proyecto
Principio 10: Dar seguimiento al plan con frecuencia y hacer los ajustes que se requieran: Los proyectos se atrasan más fácil de lo que parece, por eso es necesario revisar constantemente el avance real, compararlo con lo planificado y hacer ajustes si hay desvíos, eso evita que los problemas se acumulen y se pierda el control del proyecto.

Principios de modelado

Los modelos ayudan a entender y representar lo que se va a construir en software, desde lo que quiere el cliente hasta lo técnico, dividiéndose en modelos de requerimientos y de diseño.
Principio 1: El equipo de software tiene como objetivo principal elaborar software, no crear modelos: los modelos que ayudan a avanzar rápido son útiles, pero los que solo retrasan deben evitarse.
Principio 2: Viajar ligero, no crear más modelos de los necesarios: cada modelo nuevo consume tiempo, solo haz los que realmente faciliten el trabajo.
Principio 3: Tratar de producir el modelo más sencillo que describa al problema o al software: modelos simples son más fáciles de mantener, entender y mejorar.
Principio 4: Construir modelos susceptibles al cambio: los modelos cambiarán, pero hay que hacerlos con cuidado para no dejar cosas importantes fuera.
Principio 5: Ser capaz de enunciar un propósito explícito para cada modelo que se cree: si un modelo no tiene un objetivo claro, no vale la pena hacerlo.

Principio 6: Adaptar los modelos que se desarrollan al sistema en cuestión: diferentes sistemas necesitan diferentes tipos de modelos.
Principio 7: Tratar de construir modelos útiles, pero olvidarse de elaborar modelos perfectos: no busques perfección, sino que el modelo sirva para avanzar.
Principio 8: No ser dogmático respecto de la sintaxis del modelo: lo importante es que el modelo comunique, no que siga la notación al pie de la letra.
Principio 9: Si su instinto dice que un modelo no es el correcto a pesar de que se vea bien en el papel, hay razones para estar preocupado: confía en tu experiencia y revisa o rehace el modelo si algo no te convence.
Principio 10: Obtener retroalimentación tan pronto como sea posible: que el equipo revise los modelos para corregir errores y mejorar antes de seguir avanzando.

Resumen

Apuntes de Cátedra

Análisis y especificación de requerimientos de Software

Introducción

La etapa de análisis suele asociarse con recopilar requisitos a través de usuarios, pero estos no siempre ofrecen información precisa por falta de voluntad o dificultad para expresarse, además, pueden ocultar datos por miedo a perder privilegios o empleos, haciendo el relevamiento ineficiente por ello actualmente, se usa el término "Ingeniería de Requerimientos", que implica un enfoque proactivo para construir y validar los requerimientos en vez de solo recibirlos pasivamente. Aun así existen diferencias entre lo que el usuario quiere y lo que recibe y comprender los requerimientos es una tarea difícil porque los usuarios no siempre saben lo que necesitan y sus necesidades cambian durante el proyecto.

Ingeniería de Requerimientos

La ingeniería de requerimientos abarca todas las tareas y técnicas para entender las necesidades del cliente, adaptándose al proceso, proyecto y personas. Actúa como un puente entre el diseño y la construcción del software, desde las necesidades del negocio hasta las funciones y restricciones. Incluye siete actividades clave:
Concepción: Inicia el proyecto a partir de una necesidad o mercado, definiendo un caso de negocio y el alcance funcional, estableciendo la base para el análisis.
Indagación: Consiste en preguntar al cliente y usuarios sobre objetivos, uso y necesidades, enfrentando problemas de alcance, entendimiento y volatilidad de los requerimientos.
Elaboración: Se expande y refina la información obtenida creando modelos detallados, escenarios de usuario, clases de análisis, atributos, servicios y relaciones.
Negociación: Se resuelven conflictos entre requerimientos de distintos clientes o usuarios, priorizando y modificando requerimientos para lograr acuerdos razonables.
Especificación: Describe los requerimientos mediante documentos, modelos gráficos, prototipos o combinaciones, adaptándose al tamaño y naturaleza del proyecto.
Validación: Se revisa la calidad y consistencia de los requerimientos, corrigiendo ambigüedades, inconsistencias y errores mediante revisiones técnicas con todos los participantes.
Administración de los requerimientos: Gestiona los cambios durante todo el ciclo de vida del sistema, asegurando control y seguimiento, similar a la administración de configuración.

Origen de los requerimientos

Surgen de una necesidad de automatización, describiéndose inicialmente de forma general y luego detallándose mediante reglas de negocio y diagramas de procesos para facilitar el entendimiento técnico. El personal de sistemas transforma estos requerimientos funcionales en especificaciones técnicas para el desarrollo.

Definición de requerimiento

Según el International Institute of Business Analysis (IIBA), un requisito abarca condiciones o capacidades presentes, pasadas y futuras de la empresa, incluyendo roles, procesos, políticas y sistemas. Un requisito puede describir el estado actual o futuro de cualquier aspecto empresarial y debe especificar claramente la mejora o automatización que se desea, considerando la evolución tecnológica y el tiempo de vida de la solución.

Resumen
Bibliografía de Ian Sommerville

Requerimientos del software

Son las necesidades y restricciones de un sistema para que funcione correctamente. La ingeniería de requerimientos se encarga de identificarlos, analizarlos y documentarlos. Se dividen en requerimientos del usuario, más generales y comprensibles, y requerimientos del sistema, más detallados y técnicos, usados para el desarrollo del software. Diferentes lectores requieren distintos niveles de especificación.

Requerimientos funcionales:

Los requerimientos funcionales especifican las funciones y servicios que el sistema debe ofrecer describen cómo debe reaccionar ante diferentes entradas situaciones y acciones del usuario pueden incluir operaciones como búsqueda de información gestión de datos o interacción con otros sistemas también pueden definir lo que el sistema no debe hacer para evitar errores deben ser redactados con precisión evitando ambigüedades que puedan generar interpretaciones erróneas en el desarrollo

Requerimientos no funcionales:

Los requerimientos no funcionales no describen funciones específicas del sistema sino sus caraccteristicas generales como fiabilidad rendimiento seguridad y restricciones operativas pueden afectar la utilidad del software y su cumplimiento es clave para su correcto funcionamiento

Tipos de requerimientos no funcionales:

requerimientos del producto: especifican el comportamiento del sistema incluyen aspectos como velocidad de ejecución consumo de memoria fiabilidad y facilidad de uso

requerimientos organizacionales: surgen de políticas y procedimientos incluyen estándares de desarrollo lenguajes de programación métodos de diseño y plazos de entrega

requerimientos externos: dependen de factores fuera del sistema abarcan regulaciones legales requisitos de interoperabilidad con otros sistemas y normativas éticas

Requerimientos del dominio:

Los requerimientos del dominio provienen del area de aplicación del sistema en lugar de necesidades específicas de los usuarios incluyen términos especializados restricciones de diseño y reglas particulares pueden definir funciones nuevas modificar requerimientos existentes o establecer cálculos específicos son clave porque garantizan que el software cumpla correctamente con los estándares del sector y las regulaciones asociadas

Requerimientos del usuario:

Los requerimientos del usuario describen lo que el sistema debe hacer sin incluir detalles técnicos, deben ser comprensibles para quienes no tienen conocimientos en desarrollo se redactan en lenguaje claro evitando jerga y especificaciones de implementación su objetivo es definir el comportamiento externo del sistema, sin limitar la creatividad de los desarrolladores para ofrecer soluciones eficientes.

Requerimientos del sistema:

Los requerimientos del sistema son versiones detalladas de los requerimientos del usuario, sirven como base para el diseño e implementación del software deben ser completos y consistentes aunque en teoría no deberían incluir información de diseño en la práctica es difícil evitarlo porque se necesita definir arquitectura e interoperabilidad con otros sistemas la especificación suele redactarse en lenguaje natural pero puede complementarse con

notaciones gráficas estructuradas o matemáticas para mayor claridad.

Especificación de la interfaz

la especificación de la interfaz define cómo un sistema nuevo se comunica con los existentes debe incluirse desde el inicio del desarrollo para garantizar compatibilidad existen tres tipos principales de interfaces:

- **Interfaz de procedimientos:** permite el acceso a servicios de otros sistemas mediante la invocación de funciones o métodos se utilizan en APIs para facilitar la interacción entre programas .
- **Estructuras de datos:** organizan la información que se intercambia entre subsistemas modelos gráficos ayudan a representar estos datos y pueden ser convertidos en código automáticamente.
- **Representaciones de datos:** establecen formatos específiicos para la comunicación en sistemas embebidos como el orden de bits en protocolos de hardware se suelen describir con diagramas detallados.

El documento de requerimientos del software

El documento de requerimientos del software define oficialmente lo que los desarrolladores deben implementar, incluye requerimientos del usuario y del sistema, puede integrarlos en una única descripción o separarlos según la cantidad de especificaciones, debe equilibrar claridad para los clientes y detalle técnico para los desarrolladores, el nivel de detalle varía según el tipo de sistema y el enfoque de desarrollo, estandares como el IEEE 830-1998 establecen una estructura común pero pueden adaptarse según las necesidades del proyecto, en metodologías ágiles los requerimientos se ajustan de forma incremental para mayor flexibilidad.

Proceso de la ingeniería de requerimientos

La ingeniería de requerimientos busca definir y mantener un documento claro que especifique lo que debe hacer el sistema, el proceso incluye cuatro etapas principales, estudio de viabilidad para evaluar si el sistema es útil para el negocio, obtención y análisis donde se identifican los requerimientos necesarios, especificación para organizar la información de manera estructurada y validaciOn para garantizar que los requerimientos cumplen con las necesidades del cliente, debido a los cambios constantes en las organizaciones y tecnologías se requiere una gestión de requerimientos que permita adaptaciones en el tiempo, existen enfoques estructurados como el análisis orientado a objetos y modelos gráficos que ayudan a representar el sistema pero la obtención de requerimientos sigue siendo un proceso basado en la comunicación con los usuarios.

Estudios de viabilidad:

El estudio de viabilidad determina si un sistema nuevo es útil para la organización, evalúa si puede implementarse con la tecnología actual dentro de las restricciones de costo y tiempo, también analiza si puede integrarse con otros sistemas existentes, para ello se recopila información sobre los problemas actuales y cómo el nuevo sistema puede solucionarlos, además se consultan expertos y usuarios finales, el resultado es un informe que recomienda si continuar con el desarrollo y puede incluir ajustes en el presupuesto o alcance del proyecto.

Obtención y análisis de requerimientos:

la obtención y análisis de requerimientos es el proceso en el que los ingenieros de software trabajan con los clientes y usuarios finales para definir qué servicios debe ofrecer el sistema, su rendimiento y restricciones, involucra a diversos stakeholders que pueden tener necesidades diferentes y expresarlas en terminos poco técnicos, por lo que los ingenieros deben interpretar y negociar requerimientos para garantizar que sean biables y no entren en conflicto, este proceso es iterativo y consta de cuatro actividades principales, descubrimiento de requerimientos donde se recopilan las necesidades de los stakeholders, clasificación y organización para estructurar los requerimientos en grupos coherentes, priorización y negociación para resolver conflictos y definir lo más importante y documentación de requerimientos que recoge toda la información obtenida permitiendo refinamientos futturos.

Validación de requerimientos:

La validación de requerimientos busca garantizar que el sistema desarrollado cumpla con las necesidades del cliente, prevenir errores en los requerimientos es crucial porque corregirlos durante o después del desarrollo puede ser costoso, el proceso de validación incluye verificaciones de validez para comprobar que los requerimientos reflejan lo que realmente se necesita, consistencia para evitar contradicciones, completitud para asegurar que todas las funciones y restricciones estén definidas, realismo para verificar que sean factibles con la tecnología y recursos disponibles, y verificabilidad para permitir que cada requerimiento pueda ser probado
También se emplean técnicas de validación como revisiones sistemáticas por equipos de expertos, construcción de prototipos para que usuarios y clientes prueben el sistema antes de su desarrollo completo, y generación de casos de prueba que permiten detectar problemas antes de la implementación final, debido a la dificultad de preveer cómo el sistema encajará en el trabajo diario de los usuarios es común que los requerimientos necesiten ajustes incluso después de haber sido aprobados.

Gestión de requerimientos:

La gestión de requerimientos es el proceso de controlar y adaptar los cambios en los requerimientos de un sistema. Estos cambios surgen porque los usuarios descubren nuevas necesidades después de usar el sistema, pueden aparecer conflictos entre distintos grupos de usuarios y el entorno empresarial y tecnológico evoluciona, para manejar esto es fundamental identificar requerimientos dependientes y evaluar el impacto de cualquier modificación, se debe establecer un proceso formal para implementar cambios y asegurar que el sistema sigue cumpliendo sus objetivos, este proceso debe iniciar desde la primera versión del documento de requerimientos y mantenerse durante todo el desarrollo del software.

Actividades de aplicación

Consigna 1: Luego de haber leído comprensiva y analíticamente la bibliografía, y haber realizado el resumen reflexiona sobre estos aspectos y responde:

¿Por qué es crucial definir lo que se necesita en un software antes de desarrollarlo?

¿Cuáles son los principales desafíos en la ingeniería de software para garantizar calidad y eficiencia?

¿Cuál es la diferencia entre requisitos funcionales y no funcionales? ¿Cuál es la importancia de cada uno? Piensa en 8 ejemplos de cada uno, y relacionarlos con un software que deseen desarrollar para alguna empresa, que desee digitalizar su información.

Consigna 2: En equipos de 3-5 personas, definir los requerimientos de un software para una necesidad específica. Sigan estos pasos:

1. Seleccionen el tipo de software que van a diseñar (ejemplo: aplicación para gestionar tareas escolares).
2. Definan al menos 5 requerimientos funcionales y 3 no funcionales.
3. Incluyan 1 restricción importante para el proyecto.
4. Estructuren la información en un cuadro explicativo.

Desarrollo

Consigna 1:

Es crucial definir lo que se necesita en un software antes de desarrollarlo porque para comenzar con las etapas del desarrollo del software se requiere establecer ciertos requerimientos funcionales y no funcionales, aspectos como la preferencias del cliente, características a destacar donde se implementara, etc para poder ofrecer un desarrollo óptimo y de calidad para evitar inconvenientes en el proceso ya sea con el cliente o con los usuarios que harán uso de este software.

Los principales desafíos en la ingeniería de software para garantizar calidad y eficiencia, son los el relevamientos de requerimientos concisos sin malas interpretaciones para evitar errores en las etapas del desarrollo del software, en casos de actualizar los requerimientos ya que es inevitable ver alguna modificación de los mismos, con el tiempo, también la verificabilidad de estos es necesarias para asegurar lo solicitado por el usuario y realizar un correcto diseño del software y posteriormente su desarrollo de manera correcta.

Los requisitos funcionales serán aquellos que determinen en sí el funcionamiento principal para el cual es desarrollado y los requisitos no funcionales son para determinar cómo trabajó el software en cuanto a rendimiento de ejecución de funciones por partes de varios usuarios como tambien ltrabaja junto con la seguridad y sus restricciones.

por ejemplo:

UN usuario quiere digitalizar su empresa de gaseosas y necesita un software optimo y eficiente para trabajar rapido y de forma consistente:

Requerimientos funcionales:

- Permitir a un usuario determinado el subir informes diarios y estos que se almacenen en la base de datos para acceder cuando se requiera.
- Se requiere un sistema con jerarquías en la cual el acceso maximo de informacion sea a nombres, pedidos, gastos, etc. y lo minimos a ventas mensuales o diarias.
- Permitir un modificacion de un archivo de forma online y que se guarde de forma automatica.
- integrar software de ambitos de pagos con tarjeta de credito, tranferencias, etc.
- Permitir un vista y acceso a documentos en formatos diversos (word, excel, etc).
- proporcionar una barra de busqueda para encontrar mas rapido algunos documentos.
- Notificar eventos importantes o modificaciones de archivos importantes.
- Ofrecer reconocimiento facial en caso de datos de mayor seguridad.

Requisitos no funcionales:

- Tiempo de repuestos de 3 seg aprox. por cada proceso pesado.
- mantener conectividad 24/7.
- debera soportar grandes cantidades de documentos editandose o ejecutandose al mismo tiempo(30mil).
- que el software sea escalable.
- permitir la compatibilidad con diferentes navegadores o ply stores.
- depende ejecutarse bajo normativas especificadas por el cliente.
- optimizar almacenamiento.
- Permitir adaptabilidad a usuarios con discapacidades.