Alejandro Marotta

Proceso Unificado

Es un marco de trabajo genérico que puede ser usado para instanciar proyectos.Define el camino del desarrollo del software (Ciclo de vida) y las actividades que hay que llevar a cabo para transformar requisitos de un usuario en un sistema de software.

1. Dirigido por Casos de Uso.

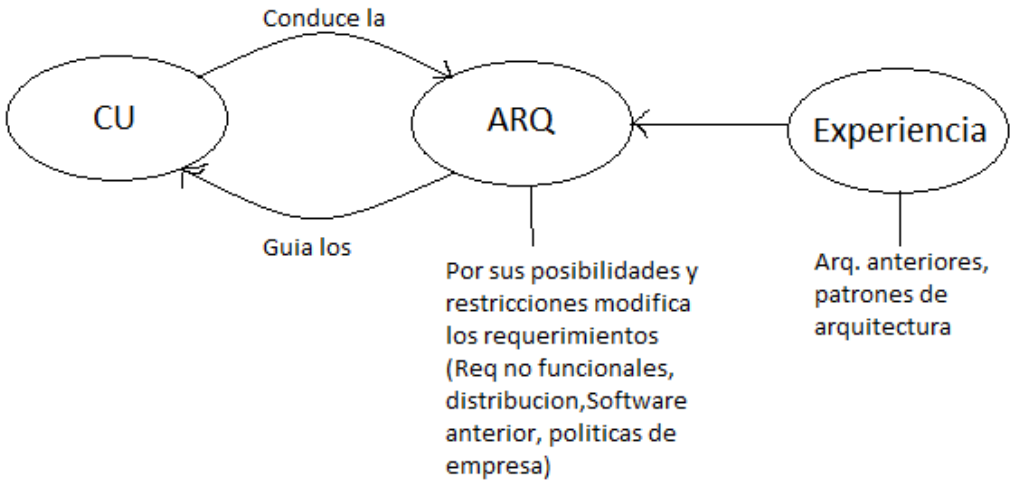
Un CU modela los requerimientos y refleja las expectativas del cliente. Representan la interacción de un actor con una funcionalidad del sistema, es decir una porción de funcionalidad que le aporta valor a un actor.
Se progresa a través de los flujos de trabajo iniciando con un conjunto de CU.

- **Permiten capturar requisitos funcionales** centrándose en el valor añadido para el usuario y no el de los programadores => centrado en el usuario.
- **Dirigen el proceso** porque dividen la complejidad de los requerimientos en porciones más pequeñas => tomo esas pequeñas partes y voy recorriendo el proceso a partir de los CU seleccionados =>todo el proceso se basa en ir tomando estas porciones.Se usan para definir el contenido de cada iteración.
- **Para idear la arquitectura** ya que los requerimientos son los que le dan la forma, también las restricciones de diseño (infraestructura preexistente, lenguaje, etc) retroalimentan y cambian los requerimientos.

Cualquier modelo o artefacto de cualquier flujo de trabajo se debe poder trazar hasta el CU que le dio sentido. Cosas que no han sido requeridas no deberían estar modeladas.
Se utilizan para conseguir un acuerdo con el cliente de lo que debería hacer el sistema.

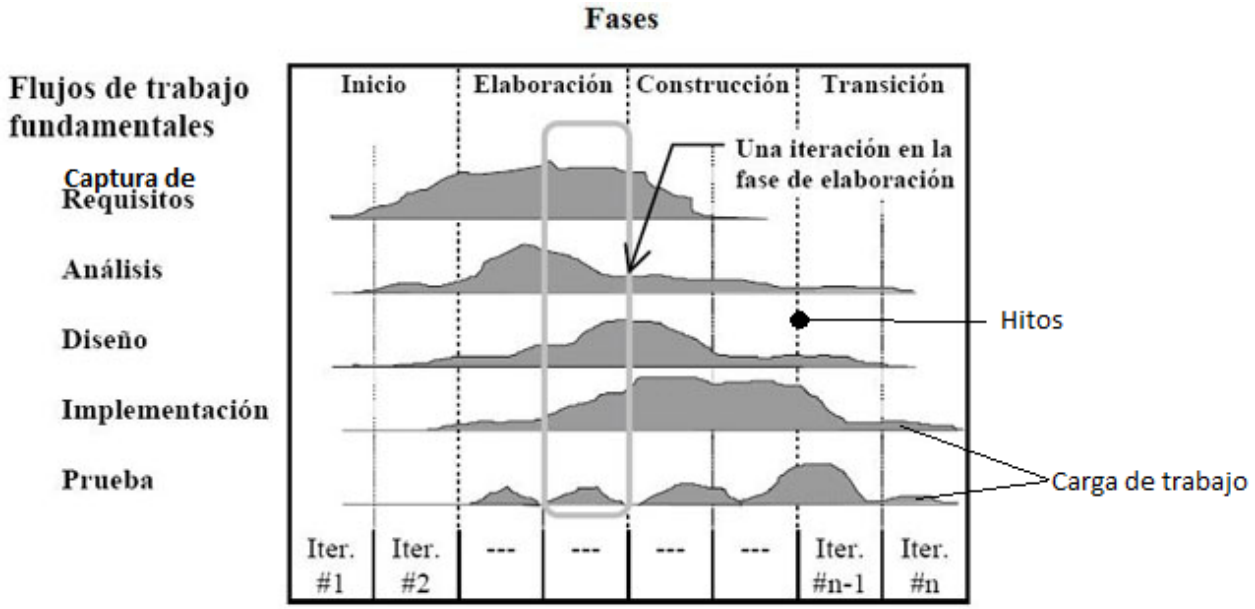
2. Centrado en la Arquitectura.

El proceso se basa en las partes más importantes del sistema y estas van a estar expresadas en la arquitectura -> parte estructural, cimientos del sistema, partes más importantes que le dan su base y luego habrán otras partes o funcionalidades que le complementan. Se representa a través de un conjunto de diagramas UML (analogía de una casa con varios planos eléctricos, agua, vigas) que dan una clara perspectiva del sistema completo. Son un conjunto de vistas (modelo de cu, diseño, implementación,despliegue) que representan las partes más importantes del modelo.
Se debe centrar en la arquitectura ya que es el esqueleto o lo fundamental del sistema => debe ser la prioridad -> si fallamos en esto lo más probable que todo falle.
Se necesita para: comprender el sistema, organizar el desarrollo, fomentar la reutilización, hacer evolucionar el sistema,etc.



3. Iterativo e incremental

Se divide el trabajo en partes pequeñas de la cual surge un incremento (crecimiento funcional en el producto). Cada fase (Inicio, Elaboración, Construcción y Transición) se divide en n iteraciones planificadas (se selecciona que CU hay que tratar, quien lo hace ,etc) y todas pasan por todos los flujos de trabajo (toma de requisitos, Análisis, Diseño,Implementación y prueba).
Ventajas: reduce el riesgo a un solo incremento, reducir la posibilidad de retrasos atacando los CU más importantes, los requisitos se pueden ir refinando en cada iteración (dificil definir todo al principio).



Un Ciclo está compuesto por 4 fases:

Inicio: Definir el objetivo (análisis del negocio) y si el proyecto es viable ej: no tenemos los recursos humanos necesarios, el tiempo no es suficiente, el cliente no tiene el suficiente dinero, la tecnología, etc. Hito de los objetivos del ciclo de vida.(Visión)
Elaboración: Defino la línea base de la arquitectura. (Al ser centrado en la arquitectura es lo primero que necesito). Con esto puedo planificar mejor las actividades y los recursos necesarios para terminar el proyecto. => todas las iteraciones de esta fase tienen que estar apuntadas a determinar la arquitectura del sistema. Hito de la arquitectura.

Línea base de la arquitectura: es la suma de todas las vistas de la arquitectura. Es un sistema pequeño que representa el esqueleto del sistema. Tiene las funcionalidades más importantes.

Construcción: Se crea el producto y la línea base de la arquitectura crece hasta convertirse en un sistema completo. Implementa todos los CU. Es una versión alfa o beta (puede no estar libre de defectos). Hito de la funcionalidad operativa o capacidad operativa inicial ->obtenemos una versión operativa.
Transición: Se prueba el producto y se informan los defectos encontrados, se corrigen los problemas y se incorporan mejoras. Siempre las iteraciones siguen agregando características al producto. Hito del lanzamiento del producto (Release).

Flujos de trabajos

Captura de requisitos

El usuario nos transmite lo que debe hacer el sistema y sus expectativas -> es una fuente imperfecta de información, además puede haber ruido en la comunicación y surgir malinterpretaciones.

Objetivos:

- Descripción del sistema correcto de lo que debe y no debe hacer el software.
- Entendible por el usuario para que pueda verificar que lo que capturamos es correcto y lo acepte o no.
- Planificación y seguimiento

=> Usamos CU como la forma en la que vamos a plasmar los requisitos y guiar el proyecto.

Pasos:

Trabajo a Realizar	Artefacto Resultante
Enumerar requisitos candidatos	Lista de características
Comprender el contexto del sistema	Modelo de dominio o de negocio
Capturar los requisitos funcionales	Modelo de Caso de uso
Capturar requisitos no funcionales	Requisitos adicionales, propiedades del sistema

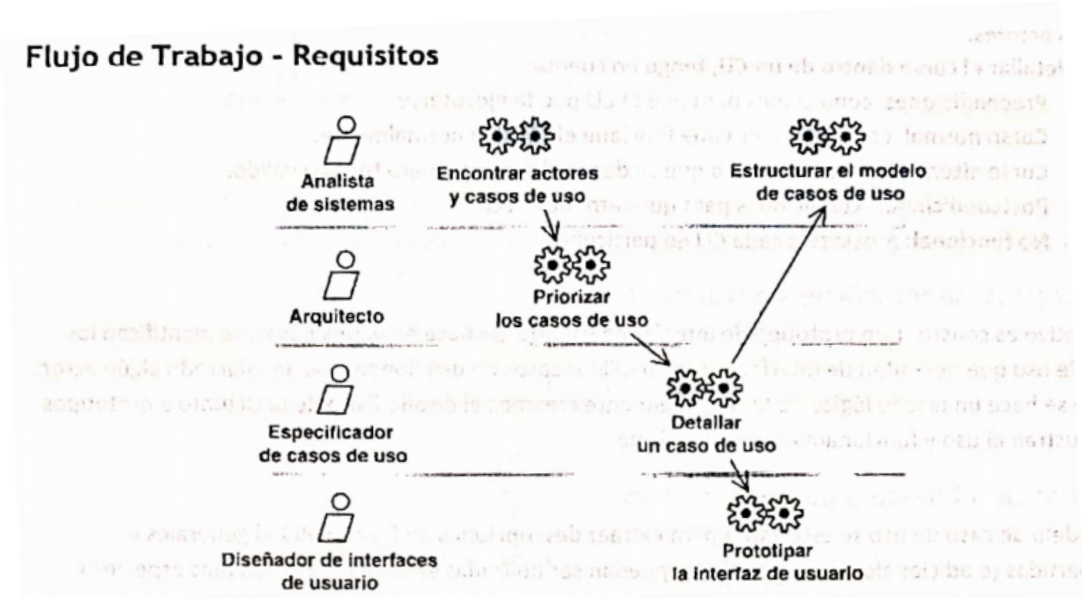


Figura 7.10. El flujo de trabajo para la captura de requisitos en forma de casos de uso, incluyendo trabajadores participantes y sus actividades.

Encontrar Actores y CU (Analista): Identificar quienes o que interactúan con el sistema, encontrar los CU, describir brevemente el objetivo de los CU, armar el modelo de CU, Definir un glosario de términos.
Entrada = Modelo de negocio - Listado de requisitos
Resultado = Modelo de CU esbozo (diseño provisional) - Glosario

Priorizar CU (Arquitecto): Identificamos los CU principales y establecemos prioridades. Basado en la arquitectura => arrancamos por los CU de mayor prioridad.(Vista de la arq).
Resultado= Descripción de la arquitectura (funcionalidad más importante y crítica).

Detallar los CU (Especificador): Describe el flujo de sucesos en detalle para cada CU (Precondiciones, curso normal,curso alternativo,postcondiciones, requisitos no funcionales).
Para formalizar la descripción y que sea más fácil de interpretar (sobre todo si tiene muchos flujos alternativos) podemos hacer diagramas de estados y de actividad.
Resultado = Detalles de CU

Prototipado de Interfaces (Diseñador): Sirve para validar que la interpretación sea correcta. Cuando el usuario lo vea puede notar si falta algo.
Resultado = Interfaces de usuario.

Estructurar el modelo de CU (Analista): Se buscan acciones comunes o compartidas por varios CU (relaciones include, extend, generalización), se eliminan redundancias, se dividen un CU muy complejo en 2, etc.
Resultado: Modelo de CU bien estructurado.

Artefactos resultantes: Modelo de dominio / de negocio, Listado de actores y CU, Diagrama de CU, Detalles de CU, prototipos de interfaces, Requisitos no funcionales, Glosario de términos.

Análisis

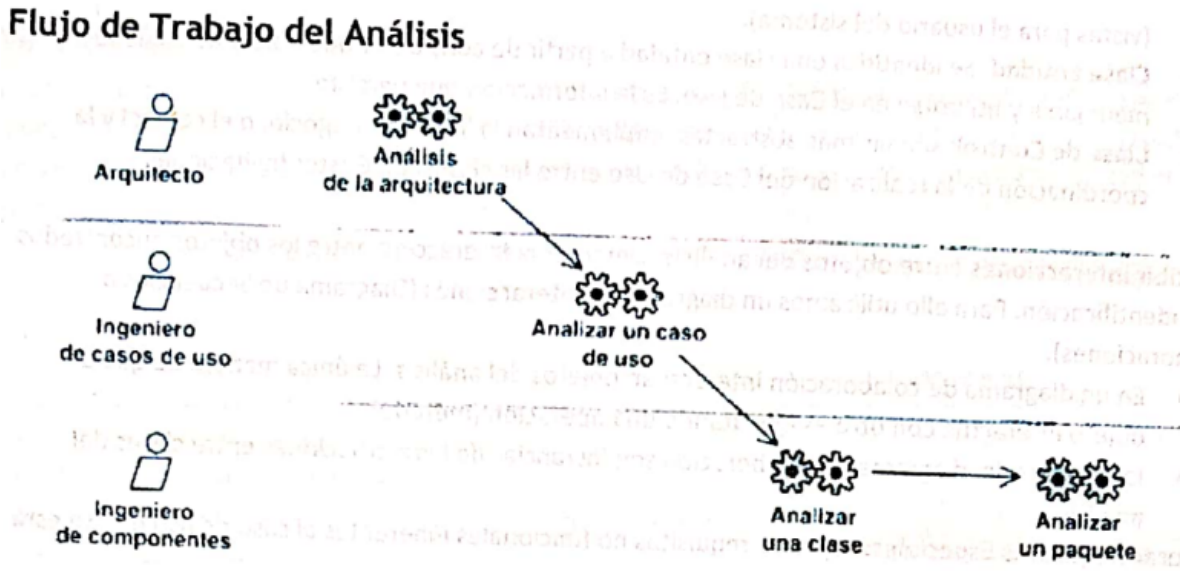
Terminar de definir qué tengo que hacer y estructurar los requerimientos, se centra en definir una arquitectura inicial de alto nivel.
Objetivos:

- Especificación más precisa de los requisitos (ambiguos, repetidos, refinarlos)
- Trasladar todo a un lenguaje más técnico (UML) -> más parecido a los que se va a programar.
- Estructurar los requerimientos haciendo abstracciones (pasamos de CU a diagramas de Clases y de Colaboración) permitiendo su comprensión y separación por partes.
- Primera aproximación al diseño. Mayor abstracción, hacemos hincapié en la funcionalidad, es decir que tiene que hacer el sistema y no tanto como lo va hacer.
- Se centra en la primera parte de la fase de elaboración.

Describe el problema

Modelo de casos de uso	Modelo de análisis
Descrito con el lenguaje del cliente.	Descrito con el lenguaje del desarrollador.
Vista externa del sistema. Como los actores interacuran con el sistema	Vista interna del sistema. Como interactuan los componentes entre si
Estructurado por los casos de uso: proporciona la estructura a la vista externa.	Estructurado por clases y paquetes estereotipados; proporciona la estructura a la vista interna.
Utilizado fundamentalmente como contrato entre el cliente y los desarrolladores sobre qué debería y qué no debería hacer el sistema. Se puede determinar como linea base de los requerimientos de que debe hacer el sistema	Utilizado fundamentalmente por los desarrolladores para comprender cómo debería darse forma al sistema, es decir, cómo debería ser diseñado e implementado.
Puede contener redundancias, inconsistencias, etc., entre requisitos.	No debería contener redundancias, inconsistencias, etc., entre requisitos.
Captura la funcionalidad del sistema, incluida la funcionalidad significativa para la arquitectura.	Esboza cómo llevar a cabo la funcionalidad dentro del sistema, incluida la funcionalidad significativa para la arquitectura; sirve como una primera aproximación al diseño.
Define casos de uso que se analizarán con más profundidad en el modelo de análisis.	Define realizaciones de casos de uso, y cada una de ellas representa el análisis de un caso de uso del modelo de casos de uso.

Esquematiza el problema desde un punto de vista más cercano al desarrollo



Análisis de la arquitectura (Arquitecto):

1. Identificación de paquetes de análisis. Estos organizan el modelo en piezas más pequeñas y manejables asociando CU dependiendo su funcionalidad.Se busca que sean altamente cohesivos (clases fuertemente relacionadas) y paquetes débilmente acoplados (poca interacción entre paquetes). También hay paquetes de servicios (notificaciones, permisos de usuario,seguridad,etc) que dan soporte a otras partes del sistema (permiten la reutilización de código). Identificar dependencias entre paquetes.
2. Identificar las clases obvias, fáciles de identificar. (Clases entidad)
3. Identificar los requisitos especiales o requisitos no funcionales (durante todo el proceso)

Analizar un CU (Ing de CU):

1. Identificar las clases del análisis del CU (entidad, control, interfaz). Esbozamos nombre, responsabilidad,atributos y relaciones. (Visión estática).
2. Describir interacciones entre los objetos del análisis (instancias de esas clases que usan métodos para interactuar) usando Diagramas de interacción (Secuencia o Colaboración) (Visión dinámica) Nota: tener en cuenta el flujo de sucesos que describimos cuando detallamos el CU en la captura de requisitos.
3. Capturar requisitos especiales o no funcionales.

Analizar una clase (Ing en componentes): Luego de identificarlos en el paso anterior:

1. Identificar sus responsabilidades (en el análisis todavía no se vuelven métodos)
2. Identificar atributos y relaciones (Asociación, agregación, generalización, cardinalidad)
3. Capturar requisitos especiales o no funcionales.

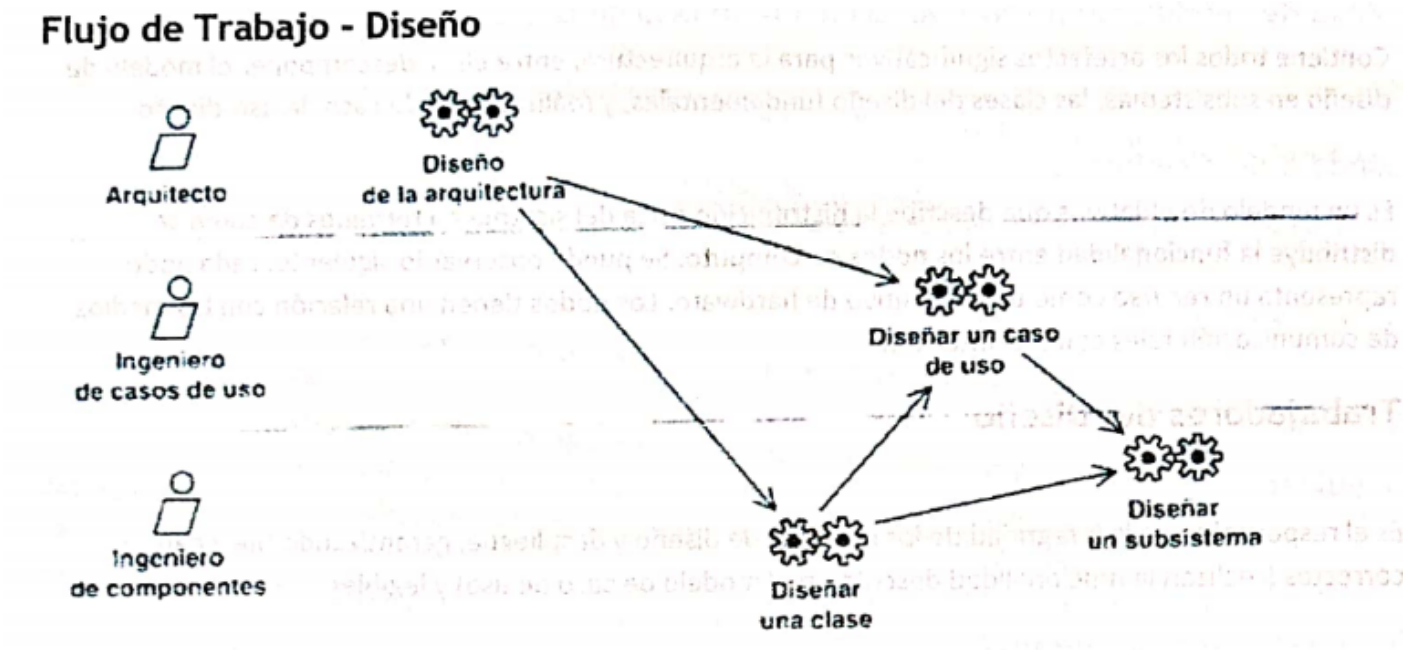
Analizar un paquete (Ing de componentes): En el paso anterior identificamos que clase colabora con cual = > de esa forma volver a analizar los paquetes para que terminen siendo lo más independiente posible -> modularidad => más fácil de mantener, modificar, reutilizar, etc. Buscamos descubrir dependencias para poder estimar los efectos de cambios futuros, asegurarnos que los paquetes tengan las clases correctas . Nunca van a ser totalmente independientes sino serían sistemas distintos.

Artefactos resultantes: diagrama de clases, de paquetes, de interacción (colaboración).

Diseño

- Modelamos el sistema para que soporte todos los requisitos (funcionales y no funcionales), SO , restricciones de tecnología, infraestructura,seguridad. Buscamos una comprensión profunda.
- Creamos un punto de partida para la implementación (los diagramas de diseño se tienen que corresponder 1 a 1 con lo que vamos a implementar cumpliendo las restricciones del lenguaje).
- Descomponer el trabajo en partes más manejables (subsistemas) => podemos programar de manera paralela distintos subsistemas. (derivan de los paquetes del análisis)
- Capturamos las interfaces entre los subsistemas (métodos públicos con los que se comunican)

Modelo de análisis	Modelo de diseño
Modelo conceptual, porque es una abstracción del sistema y permite aspectos de la implementación.	Modelo físico, porque es un plano de la implementación.
Genérico respecto al diseño (aplicable a varios diseños).	No genérico, específico para una implemetación.
Tres estereotipos conceptuales sobre las clases: Control, Entidad e Interfaz.	Cualquier número de estereotipos (físicos) sobre las clases, dependiendo del lenguaje de implementación.
Menos formal.	Más formal.
Menos caro de desarrollar (ratio al diseño 1:5).	Más caro de desarrollar (ratio al análisis 5:1).
Menos capas.	Más capas.
Dinámico (no muy centrado en la secuencia).	Dinámico (muy centrado en las secuencias).
Bosquejo del diseño del sistema, incluyendo su arquitectura.	Manifiesto del diseño del sistema, incluyendo su arquitectura (una de sus vistas).
Creado principalmente como "trabajo de a pie" en talleres o similares.	Creado principalmente como "programación visual" en ingeniería de ida y vuelta; el modelo de diseño es realizado según la ingeriería de ida y vuelta con el modelo de implemetación (descrito en el Capítulo 10).
Puede no estar mantenido durante todo el ciclo de vida del software.	Debe ser mantenido durante todo el ciclo de vida del software.
Define una estructura que es una entrada esencial para modelar el sistema —incluyendo la creación del modelo de diseño.	Da forma al sistema mientras que intenta preservar la estructura definida por el modelo de análisis lo más posible.



Diseño de la arquitectura (Arquitecto):

1. Identificamos nodos y configuración de red (conexión entre nodos, protocolos,ancho de banda) (Diagrama de despliegue)
2. Identificar los subsistemas y sus interfaces. Capas : específica de la aplicación (ej. las que manejan el tablón) , general de la app (módulos que se pueden reutilizar ej. seguridad, manejo de usuarios), capa intermedia (ej máquina virtual de java), capa del software del sistema(SO, motor de BD).
3. Identificamos las clases relevantes para la arquitectura. Tenemos las clases que representan una entidad y las clases activas (ej. controladores, servicios, configuración).
4. Mecanismos genéricos de Diseño (persistencia,concurrencia,seguridad,transacciones)

Diseñar un CU (Ingeniero de CU):

1. Identificar las clases del diseño participantes
2. Descripción de las interacciones entre objetos del diseño (Diagrama de secuencia)
3. Identificar subsistemas e interfaces participantes
4. Descripción de las interacciones entre subsistemas
5. Capturar los requisitos de implementación

Diseñar una Clase (Ing en componentes):

Definir operaciones, atributos, métodos y su descripción, relaciones, descripción de estados por los que pasa la clase, requisitos especiales (ej. concurrencia). Pensando en el lenguaje de programación.

Diseño de subsistemas (Ing en componentes):

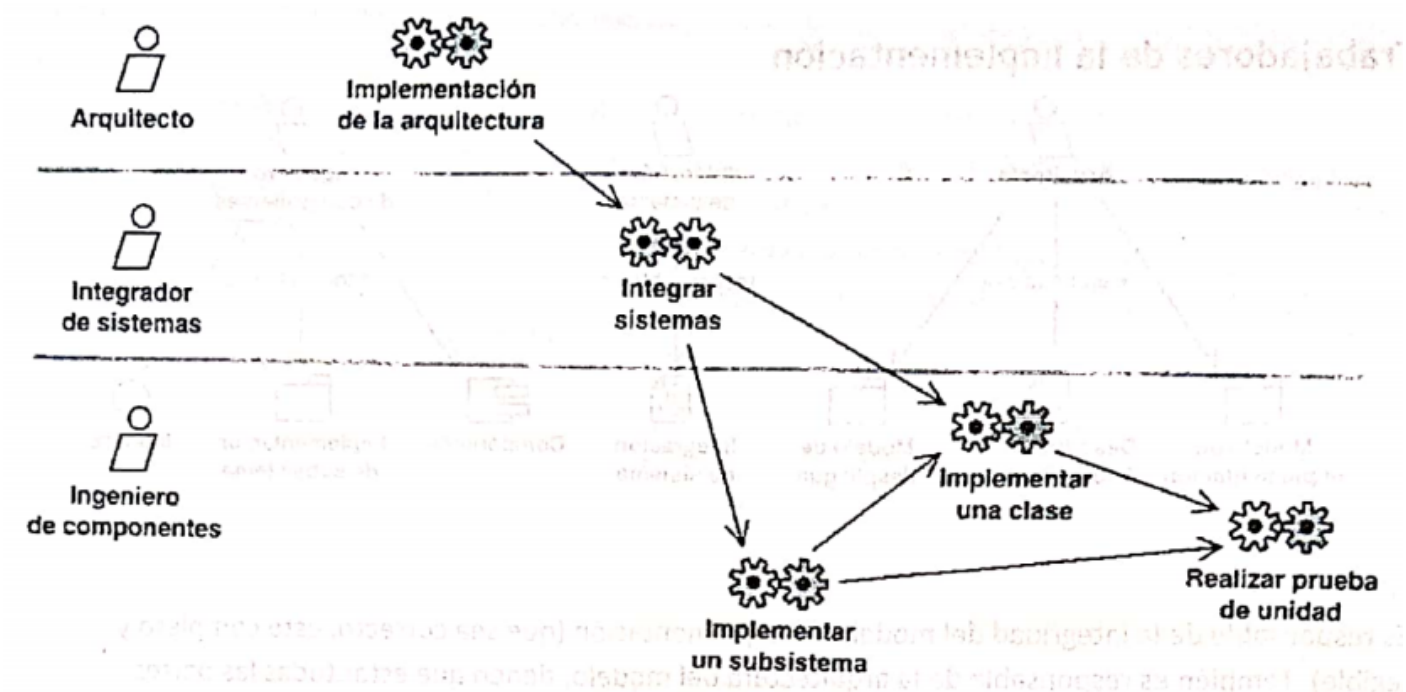
Vuelvo a analizar los subsistemas para garantizar que sean lo más independientes posible y que sus interfaces sean correctas. También veo si no surgió una nueva clase que deba incorporar.

Artefactos resultantes: Diagrama de despliegue, diagrama de subsistema, diagrama de clases, diagrama de secuencia y diagrama de estado.

Implementación

Utilizando el resultado del diseño implementamos el sistema. Se desarrolla la arquitectura y el sistema conjuntamente.

1. Planificar las integraciones del sistema por cada iteración de manera incremental.
2. Distribuir los componentes ejecutables en los nodos del diagrama de despliegue.
3. Implementar clases y subsistemas (código fuente)
4. Prueba: hacemos pruebas de unidad para cada componente y luego los integramos.



Implementación de la arquitectura (Arquitecto):

1. Identifica los componentes significativos
2. Asignar los componentes ejecutables a los nodos (según diagrama de despliegue)

Integrar el sistema (Integrador):

1. Crea un plan de integración de construcciones (describe la secuencia de construcciones necesarias en una iteración, funcionalidad que se espera que implemente y cómo probar la integración entre las partes)
2. Integración de cada construcción antes de someterlas a las pruebas de integración.

Implementar subsistema (ing en componentes):

1. Asegurar que cada subsistema cumple con su papel en cada construcción.
2. Incluir cada clase en el subsistema que corresponda
3. Comprobar que la interfaz sea la correcta (métodos públicos)

Implementar una clase (ing en componentes):

1. Genera el código a partir del diseño.
2. Se implementan las operaciones en forma de métodos.

Realizar las pruebas de unidad (ing en componentes):

1. De caja negra o de especificación: vista externa. Se ingresan parámetros de entrada y se evalúa que la salida sea la correcta.
2. De caja blanca o de estructura: vista interna. Se verifica cada sentencia ejecutada.

Artefactos resultantes: código fuente, pruebas de unidad, plan de integración de construcciones, vista de la arquitectura (elementos más representativos).

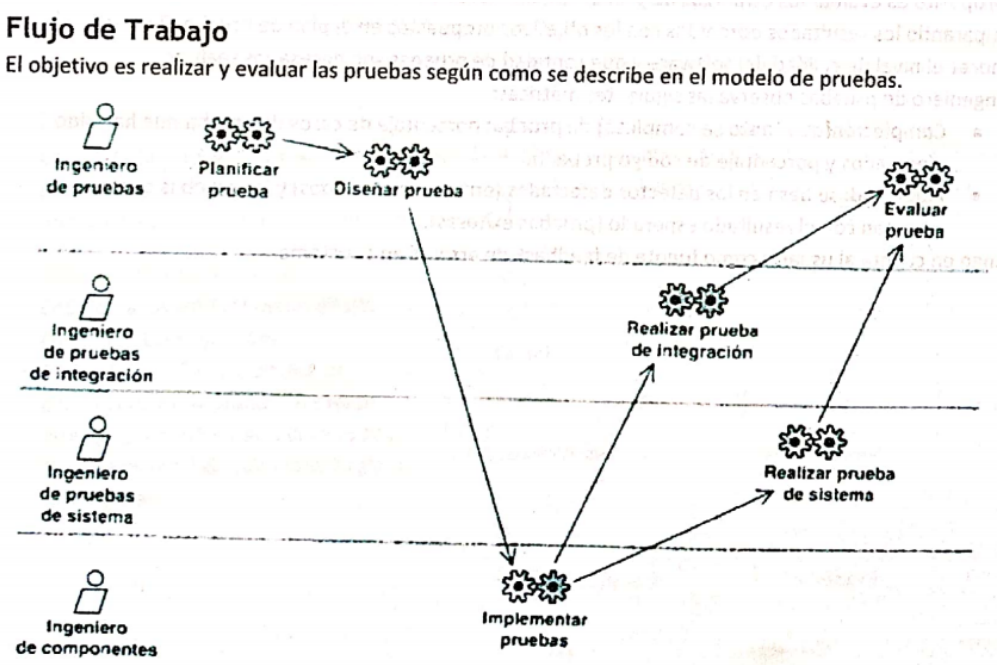
Pruebas

Verificar que las construcciones hayan sido bien implementadas.
Las pruebas del software es una investigación que nos permite obtener información acerca de la calidad del producto. Un test es exitoso cuando cubre todas las posibilidades y encuentra un fallo. Tener en cuenta que el test puede estar mal diseñado (no encuentra en error para el cual fue hecho).

Principios: Reducir el riesgo de errores, es más barato encontrar fallos. Automatizar pruebas es costoso, tardo más o menos lo mismo programando la prueba que la funcionalidad a probar. Asumir que los errores son parte del proceso, si no encuentro errores es probable que no esté testeando bien (sobre todo en las 1ras iteraciones). La etapa de prueba es una inversión.

Tipos de pruebas:

- **Unitarias** (implementación): prueban un solo componente. Son automatizables, repetibles, independientes.
- **End to End**: de caja negra, miro el flujo de datos entero de la app, desde que se cargo un dato en una pantalla hasta que recibo info de regreso.
- **Integración:** probar que todos los elementos unitarios funcionen bien en conjunto.
- **Sistema:** generalmente de caja negra. Analizan defectos globales o de comportamiento (seguridad, rendimiento)
- **Validación:** comprueba que el soft cumpla con las especificaciones (casos de prueba)
- **Exploratorias:** no tengo pruebas definidas, solo uso el sistema hasta que falle algo.
- **Rendimiento**: carga, estrés, estabilidad (ej. busquedas grandes en BD)
- **Aceptación:** pruebas con las que el Cliente aceptara el proyecto. Muy usado en metodologías ágil.



Planificar la prueba (Ing o Diseñador de pruebas): describe una estrategia de prueba y estima el esfuerzo necesario. Tanto de integración como las de sistema. Hay que buscar pruebas representativas (ej. solo lo pruebo en Chrome y Edge actuales).
Diseñar prueba(Ing de pruebas): Describir los casos de prueba (Objetivo, estado inicial, datos, resultado esperado). Se busca que las pruebas sean repetibles, es decir poder seguir la misma secuencia de pasos para llegar al mismo error (reproducir errores para saber que hay que arreglar).
Implementar prueba(Ing en componentes): Se automatizan los procedimientos que sean posibles.
Realizar pruebas de integración (Ing en pruebas de integración)
Realizan pruebas de sistema (Ing en pruebas de sist): Estos pueden empezar solo cuando las de integración tienen éxito.
Evaluar las pruebas(Ing en pruebas): Se evalúan los resultados comparando los resultados obtenidos con los objetivos propuestos en el plan de pruebas para ver la calidad del software.
(Compleción->porcentaje de código probado, Fiabilidad-> se basa en los defectos encontrados y las pruebas exitosas)

Artefactos resultantes: Plan de prueba (CU a testear,funcionalidades a probar,herramientas,Alcance, Ambiente de prueba, criterios de éxito), procedimiento de prueba (pasos), casos de prueba, resultado de las pruebas.

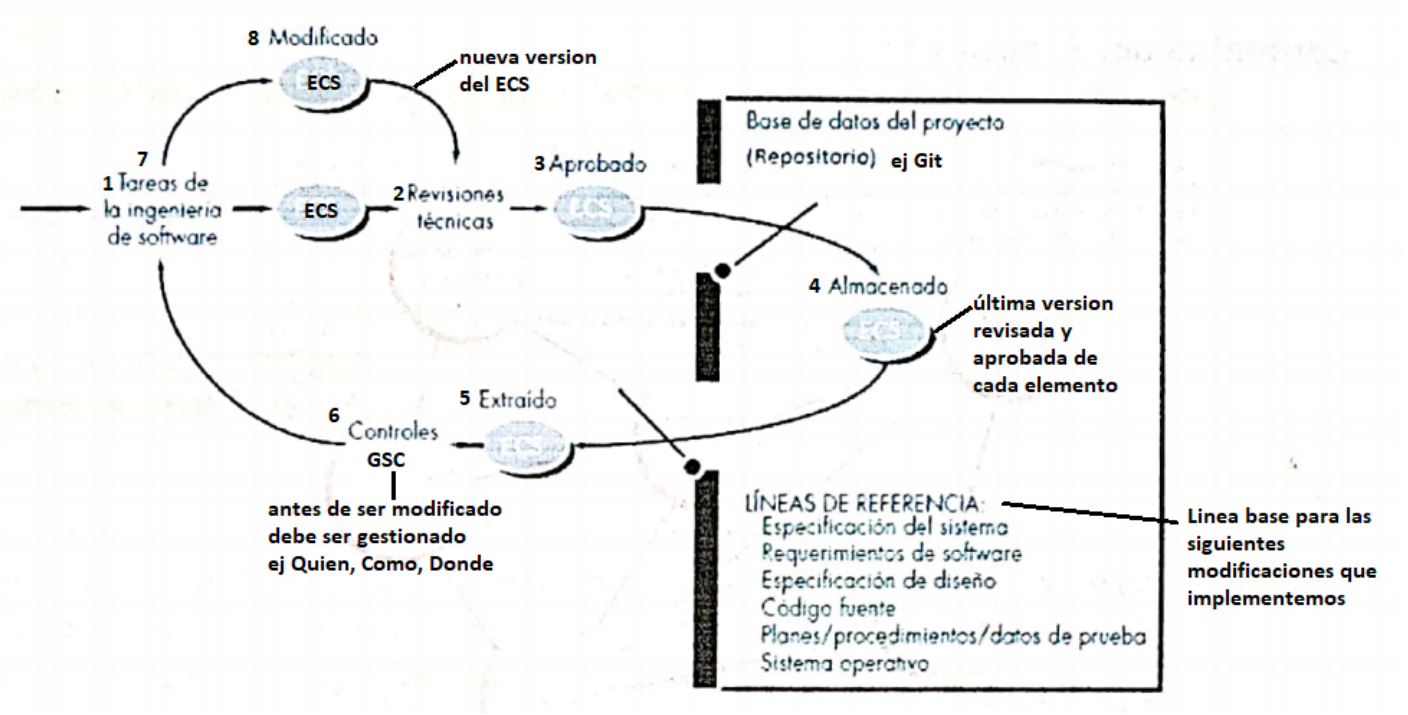
Gestión de la Configuración del Software

Configuración: Disposición de las partes que componen una cosa y le dan su forma y propiedades. Conjunto de cosas que forman algo.
Elementos de configuración del software (ECS): cualquier tipo de información resultante de los procesos de ingeniería de software (Ejecutables, Código, documentación, Datos)
=> **Configuración de Software :** es el conjunto de todos los ECS.(toda la info producida)

La Gestión es un conjunto de actividades diseñadas para controlar el cambio. Es una actividad de protección del proceso que reduce los riesgos. Sirven para identificar el cambio, controlarlo, implementarlo adecuadamente e informar a las partes interesadas. Ya que no importa en qué punto del ciclo de vida estemos, los cambios son inevitables.

Orígenes de los cambios: Cambios en los requisitos(nuevas necesidad del cliente o por el mismo avance del proceso), condiciones del mercado, reorganización de la empresa (cambio de políticas), restricciones presupuestarias o de calendario.

La gestión es compleja no solo porque hay muchos ECS, sino también porque el cambio es muy dinámico => hay que encararlos de manera profesional y gestionarlo correctamente. Sino puede causar caos y confusión.



1. Cualquier tarea que genere un ECS (toma de requisitos, Análisis ,etc)
2. Luego de generar un ECS tengo que revisarlo y aprobarlo por distintos procedimientos dependiendo el artefacto. Ej CU se los muestro al cliente, en la implementación de test unitarios.

Línea base : es un producto que se revisó formalmente y se aprobó y que todos están de acuerdo, sirve como base para continuar el desarrollo y solo puede cambiarse a través de procedimientos de control de cambios formales. Es una línea de referencia.

Objetivos:

- Identificar los ECS que pueden cambiar.
- Definir mecanismos para administrar distintas versiones y controlar los cambios (herramientas como Git ->CSV , SVN).
- Garantizar que el cambio se implemente de manera adecuada implementando procedimientos o las políticas adecuadas(serie de pasos que se tienen que llevar a cabo para que un cambio sea realizado y aprobado).
- Auditar los cambios (asegurar la calidad revisando si se hicieron bien las modificaciones, se siguió el procedimiento adecuado etc).
- Generar reportes e informar al equipo.

Métricas

Medida: proporciona una indicación cuantitativa de algún atributo del producto o proceso (líneas de código, tiempo entre fallo y fallo, tiempo en el que tarda una tarea).
Medición: acto de tomar una medida
Métrica: combinación de tomar varias medidas, combinarlas y tratar de obtener un indicador que proporcione una visión de cómo está el proceso de desarrollo o el producto, saber donde estoy parado. Sirve para poder ajustar las cosas y que salgan mejor.

Objetivos:

- Indicar la calidad de un producto. Aseguramiento de la calidad, se enfoca en el proceso de desarrollo.
- Hay que controlarlo bien y tener mejora continua. Para saber si estamos cumpliendo con los estándares de calidad hay que medir. Ej. medir cuánto me tardo en especificar un CU para ver si cumplo con el calendario o una medida técnica sobre el producto puede ser fiabilidad (fallos/tiempo).

- Evaluar la productividad de la gente. ej velocidad en la que trabajan en ciertas circunstancias para poder estimar tiempos futuros.
- Establecer una línea base para las estimaciones (las medidas pueden usarse en futuras etapas o proyectos para hacer estimaciones más precisas)
- Justificar herramientas necesarias o formación o capacitaciones adicionales.

Tipo de medidas: Directas (costo, cantidad de líneas de código, tiempo,) o indirectas (usabilidad, mantenibilidad,escalabilidad) que pueden llegar a ser subjetivas => hay que realizar un proceso bien definido y acordado para llegar a una conclusión.

Actividades:

1. Formulación: para que voy a medir, que voy a medir, cual es el costo-beneficio. No voy a medir todo sino que tengo que planearlo. Definir un Objetivo (ej calidad -> mido fiabilidad, escalabilidad, mantenibilidad). Preguntarme qué tengo que medir para cumplir el objetivo, como lo mido? . Y qué métricas voy a calcular.
2. Recolección de datos (medir)
3. Análisis de los datos: calcular las métricas
4. Interpretación: Evaluar los resultados
5. Retroalimentación

Clasificación de métricas: Productividad (asociadas al proyecto de desarrollo y estimaciones), de Calidad y Técnicas (asociadas al producto).

Deberían: Ser simples y fáciles de usar, es mejor tener pocas cosas y que sean indicadores útiles antes que medir por medir, ser consistentes y objetivas, no utilizar combinaciones extrañas de unidades, independientes del lenguaje de programación y la tecnología.

Calidad

Es un atributo de un producto o servicio. Es subjetivo para el cliente.
Para nosotros: “Conformidad de las expectativas del Cliente”
Calidad del Software - 3 puntos fundamentales a cumplir

1. Concordancia con los requisitos ->expresan las expectativas del cliente. Tanto funcionales como no funcionales. Hay que encontrar un equilibrio entre lo que ve y no ve el cliente.
2. Cumplimiento de los requisitos implícitos. Ejemplo que no ande lento, que no se trabe. Seguridad.
3. Seguir los estándares especificados por nosotros y por el cliente Ej. Documentar, comentar, definir cómo se declaran las variables, sintaxis, que todos hagan las tareas igual, definir una metodología de trabajo (ejemplo Scrum), cómo se va probar y mantener el soft. Todos los estándares tienen que estar especificados y documentados para que todos puedan seguirlos.

Evolución:

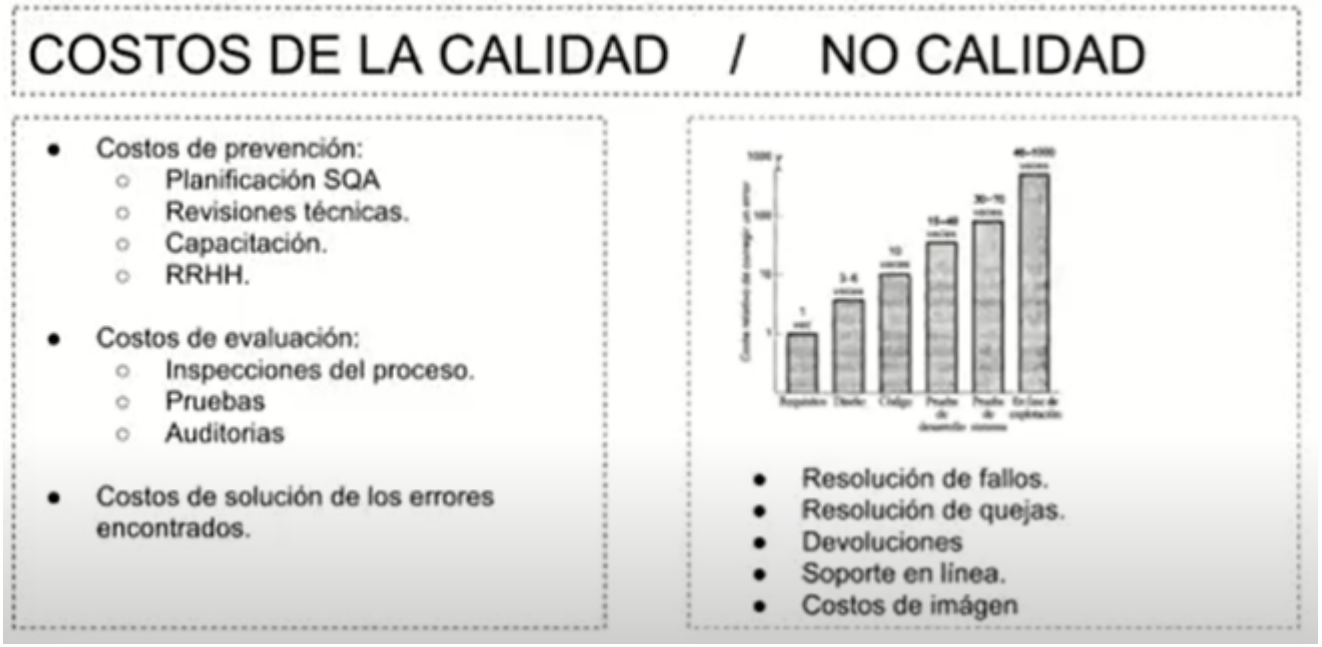
1. Control de calidad. Reducir la variabilidad de un producto a otro. ej que todos los tornillos sean iguales. No aplica tanto al software.
2. Aseguramiento de la calidad: no controla solo el producto final sino que la aseguro durante todo el desarrollo y proceso de producción. (ISO 9001, CMMI). Pensados para cumplir con todos los requerimientos implícitos que hay que cumplir.
3. Calidad Total: no solo cumplir las expectativas del cliente respecto al producto, sino tambiéntodo lo que mi organización brinda alrededor. ej. soporte,atención al cliente.

Aseguramiento de la calidad

- Descripción del proceso de desarrollo -> bien documentado. Caminos a seguir repetibles y medibles para ser mejorados.
- Revisión de las actividades de software ->revisiones técnicas de la línea base, revisión de codigo con test
- Asegurar que todas las desviaciones del proceso se documenten ej saltarme un paso por falta de tiempo
- Obtener feedbak para la mejora continua -> aprender de los errores

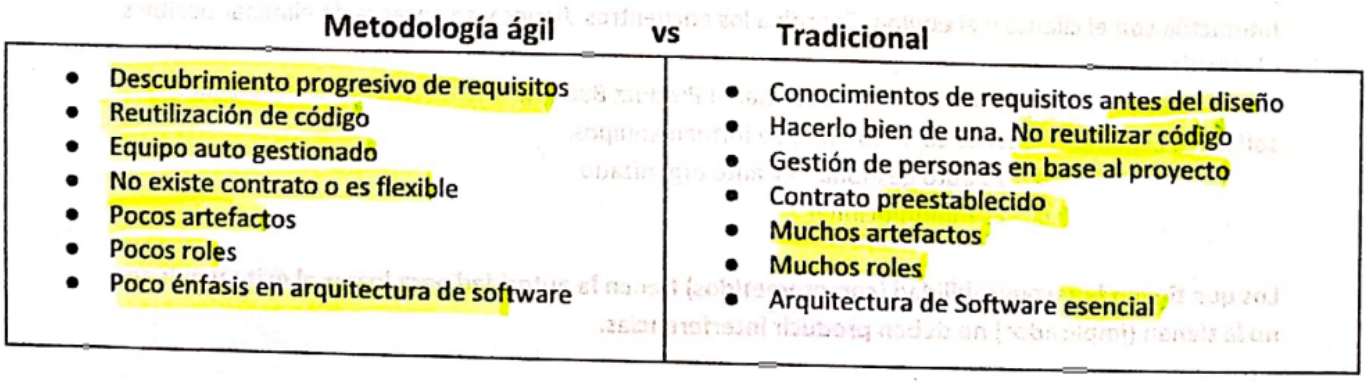
Alejandro Marotta

Revisión de software
El objetivo es encontrar errores durante el proceso de desarrollo => nunca se transforma en un fallo o defecto encontrado por el cliente que es de 10 a 10000 veces más caro.
Cultura de no castigar a la persona que cometió el error, el error es humano => permite que el proceso mejore.
Limitar el debate, la idea es encontrar el error, luego habrá otras instancias donde se arreglan.



Es más barato encontrar errores en etapas tempranas y la NO calidad no me permite esto.

Metodologías Ágiles



Podemos definir las metodologías ágiles como un conjunto de tareas y procedimientos dirigidos a la gestión de proyectos (marco de trabajo). Busca solucionar los problemas de calidad insuficiente y que el proyecto no se exceda en tiempos y costos, minimizando los riesgos que pueden surgir de desarrollos extremadamente largos. Busca maximizar los beneficios.

Existen 4 principios fundamentales:
- Individuos e interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.
- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio sobre seguir un plan.

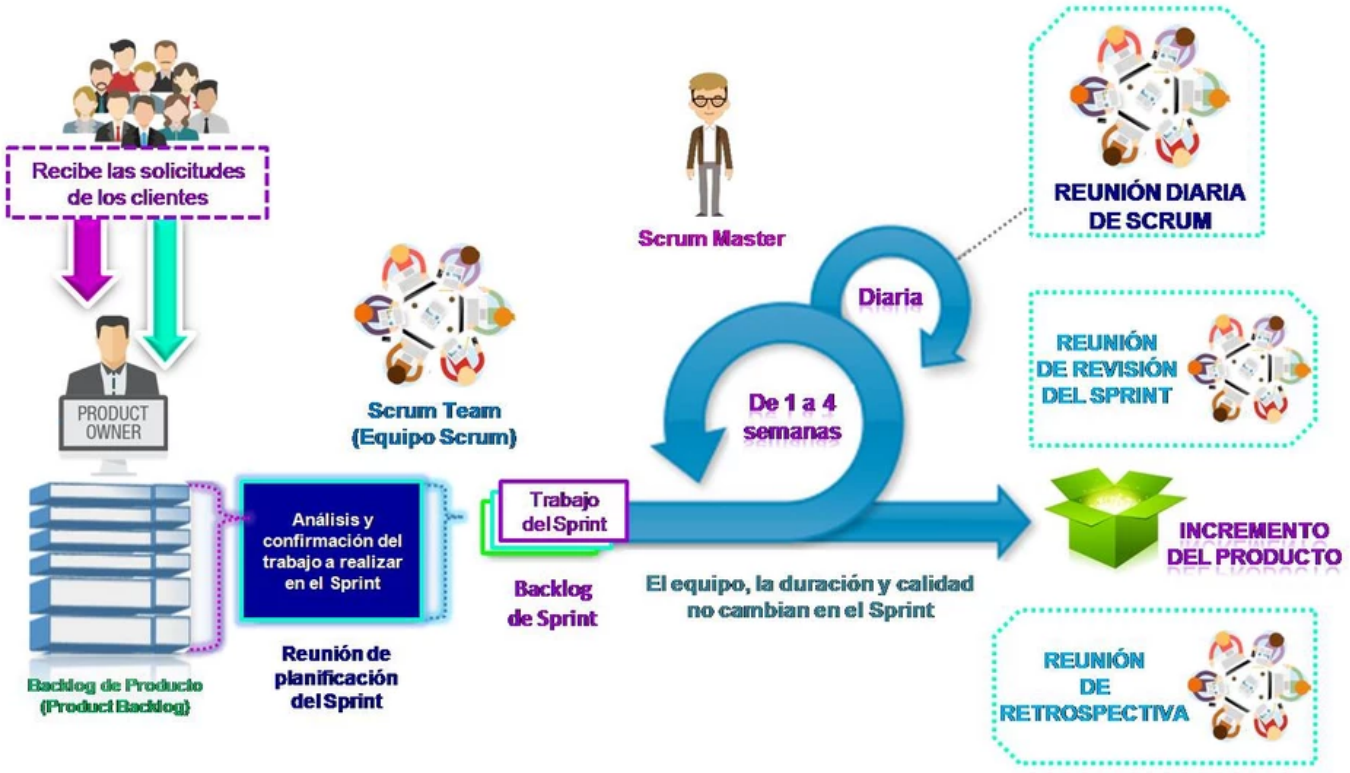
Esto lo logramos por ejemplo satisfaciendo al cliente haciendo entregas periódicas, siendo flexibles al cambio, el proceso de desarrollo está guiado por el cliente, dialogo constante entre el equipo, producir código claro y autodocumentado, etc

Alejandro Marotta

Scrum

Trabaja con un ciclo de vida iterativo e incremental, donde se va liberando el producto de forma periódica, aplicando las buenas prácticas de trabajo en equipo. Busca eliminar la burocracia para producir resultados en cortos periodos de tiempo.

Stakeholder: Es el cliente, su responsabilidad radica en definir los requerimientos (Product Backlog), recibir el producto al final de cada iteración y proporcionar el feedback correspondiente.
Product Owner: Es el intermediario de la comunicación entre el cliente (stakeholder) y el equipo de desarrollo. Es el que entiende las reglas de negocio y debe priorizar los requerimientos según sean las necesidades de la solicitud.Debe definir los criterios de aceptación.
Scrum Master: Actúa como facilitador ante todo el equipo de desarrollo, elimina todos aquellos impedimentos que identifique durante el proceso, así mismo se encarga de que el equipo siga los valores y los principios ágiles, las reglas y los procesos de Scrum, incentivando al grupo de trabajo.
Scrum Team (Equipo de desarrollo): Se encarga de desarrollar los casos de uso definidos en el Product Backlog, es un equipo auto gestionado lo que quiere decir que no existe un de jefe de equipo, irán construyendo los Sprint Backlog. Son un grupo interdisciplinario.



Programacion Extrema

Es una metodología basada en buenas prácticas para el desarrollo de software en ambientes muy cambiantes con requisitos imprecisos, por ende está enfocada en la retroalimentación continua entre el equipo de desarrollo y el cliente.Es por ello que iniciando el proyecto se deben definir todos los requisitos, para luego invertir el esfuerzo en manejar los cambios que se presenten y así minimizar las posibilidades de error. XP tiene como base la simplicidad y como objetivo la satisfacción del cliente.

- Desarrollo iterativo e incremental.
- Programación en parejas.
- Pruebas unitarias continuas.
- Corrección periódica de errores.
- Integración del equipo de programación con el cliente.
- Simplicidad, propiedad del código compartido y refactorización del código.

En el flujo de diseño se usan tarjetas CRC (Clase,Responsabilidad,Colaboración), que es un mecanismo sencillo y eficaz para describir el software. Son el único artefacto resultante del diseño.

Patrones de Diseño

Solución genérica a un problema conocido y recurrente. => ser un concepto aprobado y describir los participantes y sus relaciones. Son buenas prácticas de programación.

Estructura: Nombre, Clasificación(creacional, estructural, comportamiento), intención (que busca solucionar), motivación (cuál fue el origen por el que se creó) , estructura(diagrama de clase), Colaboraciones (Diag de secuencia), Consecuencias (ejemplo lentitud), Implementación (ejemplo en código).

- Creacional: se encarga de crear objetos (Singleton, Factory)
- Estructural: componer clases o ensamblar objetos (Facade,Composite)
- Comportamiento: describen flujos de control (Observer , State ->permite cambiar el comportamiento cuando el estado interno del objeto cambia).