TEMA 088. ANÁLISIS FUNCIONAL DE SISTEMAS, CASOS DE USO E HISTORIAS DE USUARIO. METODOLOGÍAS DE DESARROLLO DE SISTEMAS. METODOLOGÍAS ÁGILES.

Actualizado a 20/04/2023

1. ANÁLISIS FUNCIONAL DE SISTEMAS, CASOS DE USO E HISTORIAS DE USUARIO

El **análisis del sistema es** una actividad crucial en el proceso de desarrollo de software que busca entender las necesidades y restricciones del cliente y definir un conjunto de requisitos para crear un diseño.

Las **tareas** incluyen evaluar la viabilidad técnica, económica y legal del sistema, asignar funciones y compromisos a los elementos del sistema y especificar los requisitos del sistema.

1.1. ANÁLISIS FUNCIONAL DE SISTEMAS

Según SWEBOK, consiste en identificar los requisitos del mismo, describiendo QUÉ se va a desarrollar. Los principales pasos que se llevan a cabo durante la ingeniería de requisitos son:

- **Educción, identificación o extracción de requisitos**.
- **Análisis de requisitos**.
- **Representación de requisitos**.
- **Validación de Requisitos**.

Para reducir los problemas y la complejidad → emplear técnicas: **Abstracción**, **Partición y Proyección**.

Fase de Requisitos del Software se puede descomponer en dos actividades:

- **Análisis de requisitos del software → Documento de Requisitos del Sistema (DRS).**
- **Especificación funcional o Descripción del Producto**, **Documento de Diseño Funcional (DDF**).

Ambos documentos conforman el **Documento de Especificación de Requisitos Software (ERS)**.

Según Sommerville, los requisitos pueden clasificarse en: **Requisitos de sistema** y **Requisitos software**.

Las estrategias y técnicas de educción se clasifican en:

- **De alto nivel:** JAD, JRP, Entorno de Bucles Adaptativo, Prototipos, FCE, Historias de usuario.
- **De bajo nivel:** Entrevistas, Brainstorming, PIECES. Casos de Uso, Análisis de Mercado.

1.2. CASOS DE USO

El modelado de casos de uso es una técnica de obtención de requisitos que resulta adecuada para sistemas que están dominados por requisitos funcionales del usuario. Pasos que se llevan a cabo: 1. Determinar el sujeto o **límite del sistema**, 2. Encontrar **actores**, 3. Encontrar **casos de uso** y 4 Refinar hasta obtener un límite claro para el sistema.

Permite capturar los requisitos funcionales y expresarlos desde el **punto de vista del usuario**.

Elementos que forman parte del diagrama de Casos de Uso:

- **Actores:** Entidad externa al sistema que realiza algún tipo de interacción con el mismo.
- **Caso Uso:** Secuencia acciones realizada por Sistema que produce resultado. Requisito Funcional.
- **Relaciones entre actores y casos de uso:** de comunicación o solicitud.
- **Relaciones entre casos de uso: <<extiende/extend>> y <<usa/include>>**.
- **Contexto del sistema:** Cuadro que contiene las diferentes partes del sistema y los casos de uso.

Generalización

- **Generalización de actores**: cuando tenemos actores que heredan roles de su/s actor/es padre.
- **Generalización de casos de uso**: los CU hijos heredan las características del caso de uso padre.

1.3. HISTORIAS DE USUARIO

- Siguen la plantilla, según Mike Cohn: **COMO <rol> QUIERO <algo> PARA PODER <beneficio>**
- Las grandes historias de usuario se conocen como épicas (**epics**).
- Se componen de **3 partes** (conocidas como Card, Conversation, Confirmation).
- **Características** (INVEST): Independent, Negociable, Valuable, Estimable, Small y Testable.
- **Criterios aceptación** → SMART: Specific, Measurable, Achievable, Relevant y Time-bound.

Existen dos técnicas para escribir criterios de aceptación:

- Técnica de comportamiento: DADO <condiciones> CUANDO <evento> ENTONCES <resultado>.
- Técnica de escenarios: Suele definir "el trayecto feliz" y un trayecto alternativo.

Historias de usuario	Casos de uso
Más cortas, estimables (entre 10h y 2 semanas)	**Más largos**. En ocasiones difícil dar estimación.
Se **pueden ir refinando y completando**..	Debería **estar completa desde el principio**.
Se suelen desechar una vez finaliza la iteración.	Perviven en el tiempo
Están **incompletas sin las pruebas de validación.**	
Escenarios alternativos -- diferentes condiciones	Escenarios alternativos -- flujos alternativos.
Pueden especificar **requisitos no funcionales.**	**Siempre se refieren a requisitos funcionales**

2. METODOLOGÍAS DE DESARROLLO DE SISTEMAS

Metodología: Conjunto de **procedimientos**, **métodos o técnicas** y **herramientas** con un **soporte documental** que ayuda a los desarrolladores a realizar nuevo software. Algunos tipos de metodologías:

- Para Sistemas de Tiempo Real. son ampliaciones y modificaciones de otras metodologías.
- Estructuradas. Orientadas a datos (o función) o procesos. SSADM, Merise, Metrica 3, etc.
- Metodologías OO.. OOAD – BOOCH, OMT-Rumbaugh, OOSE- Jacobson → **RUP** o PUDS.
- Metodologías para un Dominio Específico. Por ejemplo, herramientas CASE y BPM.
- Metodologías Ágiles. Scrum, XP, Kanban, FDD…

Metodología vs Ciclo de Vida → una metodología puede seguir uno o más ciclos de vida.

3. METODOLOGÍAS ÁGILES

Manifiesto Ágil, se desarrollan los 12 Principios del Manifiesto Ágil

1. Principal prioridad **satisfacer al cliente** → entrega temprana y continua de software de valor.
2. Son **bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo**.
3. **Entrega frecuente software que funcione**, en periodos de 2 de semanas a 2 meses,.
4. **Las personas del negocio y los desarrolladores deben trabajar juntos** de forma cotidiana.
5. Construcción de proyectos en torno **a individuos motivados**.
6. La forma más eficiente y efectiva de comunicar información → **conversación cara a cara**.
7. El **software que funciona** es la principal medida del progreso.
8. Se debe mantener un **ritmo constante de forma indefinida**.
9. **La atención continua a la excelencia** técnica enaltece la agilidad.
10. La **simplicidad** como arte de maximizar la cantidad de trabajo que no se hace, es **esencial.**
11. **Las mejores arquitecturas, requisitos y diseños emergen de** equipos que se auto-organizan.
12. En intervalos regulares, el **equipo reflexiona** sobre cómo ser más efectivo y ajusta su conducta.

3.1. SCRUM

SCRUM (*melé*, en Ruby) es un conjunto de prácticas y recomendaciones para el desarrollo de un producto, basada en un proceso iterativo e incremental y que promueven la comunicación, el trabajo en equipo y el feedback rápido sobre el producto construido. Sus objetivos principales son:

- Mostrar producto construido con frecuencia para obtener feedback por parte del usuario mediante entregas tempranas y planificadas.
- Conseguir equipos de alto rendimiento.

Roles El conjunto de roles participantes se divide en: cerdos (los comprometidos) y gallinas (implicados).

- Product Owner.
- ScrumMaster.
- ScrumTeam o Equipo.

Otros elementos

- Product Backlog.
- Sprint.
- Burn down Chart.
- Product Backlog Items.
- Sprint Backlog.
- Reuniones: Sprint Planning Meeting, Daily scrum, Sprint Review y Sprint Retrospective.

3.2. KANBAN

Kanban se basa en un sistema de producción que dispara trabajo solo cuando existe capacidad para procesarlo. Es una aproximación al proceso gradual, evolutivo y al cambio de sistemas para las organizaciones. Un 'tablero' simple consistiría en 3 columnas: To-Do, Doing y Done.

Demuestra ser una de las metodologías adaptativas que menos resistencia al cambio presenta.

Podemos destacar 4 principios del método:

1. Comience con lo que hace ahora: estimula cambios continuos, incrementales y evolutivos.
2. Persigue el cambio incremental y evolutivo, mediante pequeños y continuos cambios.
3. Respetar el proceso actual, los roles, las responsabilidades y los cargos.
4. Liderazgo en todos los niveles..

Herramientas: tableros. Kanban Tool, JIRA, Greenhopper, Trello, Cardmapping o Tablero Kanban online

3.3. XP (EXTERME PROGRAMMING)

Adecuada para proyectos con requisitos cambiantes y equipos pequeños o medianos. Se basa en la retroalimentación continua del cliente y el equipo de desarrollo, con el objetivo de reducir costos mediante la estimación de la velocidad del proyecto. Los proyectos comienzan obteniendo User Stories y desarrollando soluciones Spike Solutions sobre una idea arquitectural general de la solución conocida como Architectural Spike.

3.4. FDD (FEATURE DRIVEN DEVELOPMENT)

Se centra en iteraciones cortas que entregan funcionalidad tangible. Conveniente para el desarrollo de sistemas críticos. FDD basa en funcionalidades pequeñas del producto. Consta de los siguientes pasos:

- Análisis y Desarrollo de un modelo del producto.
- Construir una lista de funcionalidades.
- Planificación basada en las funcionalidades identificadas.
- Diseño basado en las funcionalidades, realizado en iteraciones.
- Construcción basada en las funcionalidades, realizado en iteraciones.

3.5. TDD (TEST DRIVEN DEVELOPMENT)

Promueve los test unitarios de cada una de las nuevas funcionalidades. Involucra otras prácticas (ciclo): **Escribir las pruebas primero (Test First Development)** y **Refactorización (Refactoring).**

ATDD (Acceptance Test Driven Development) técnica en la cual se espera que el usuario final defina y diseñe las pruebas que den el OK a su uso. También se le llama STDD - Story Test Driven Development.

3.6. BDD (BEHAVIOR DRIVEN DEVELOPMENT)

Surgió a partir del desarrollo guiado por pruebas. El corazón del BDD es la reconsideración de la aproximación a la prueba unitaria y a la prueba de validación.

3.7. DDD (DOMAIN DRIVEN DESIGN)

Busca conectar la implementación del software con los conceptos y modelos del negocio para abordar necesidades complejas. Su premisa principal es centrarse en la lógica y el núcleo del dominio, basándose en un modelo para los diseños complejos. DDD también promueve una colaboración creativa entre los técnicos y los expertos del dominio para lograr una comprensión más profunda de las necesidades del negocio.

3.8. LEAN

Se enfoca en la entrega continua de valor al cliente y la mejora constante del proceso de desarrollo, eliminando el desperdicio y aumentando la eficiencia y calidad. Los siete principios de la metodología LEAN en el desarrollo de software buscan aumentar la eficiencia, reducir costos y mejorar la calidad.