Resumen de la materia Ingeniería del Software I

Agustín Curto, agucurto95@gmail.com
Francisco Nievas, frannievas@gmail.com

2017

Contents

1 Introducción **3**

2 Requerimientos de Software **5**
2.1 Análisis del problema o requerimientos. 5
2.2 Especificación de requerimientos: 6
2.3 Validación 7
2.4 Métricas 7

3 Arquitectura del software **8**
3.1 El rol de la arquitectura 8
3.2 Vistas de la arquitectura 10
3.3 La vista componentes y conectores 10
3.4 Estilos arquitectónicos para la vista C&C 10
3.5 Tubos y filtros 10
3.6 Datos compartidos 11
3.7 Estilo cliente-servidor 11
3.8 Estilo publicar-suscribir 11
3.9 Estilo peer-to-peer 11
3.10 Estilo de procesos que se comunican 11
3.11 Relacion/diferencia entre diseño y arquitectura 12
3.12 Preservación de la intergridad de la arquitectura 12
3.13 Método de análisis ATAM 12

Nota: Este resumen se corresponde con la materia dictada en el año 2017. Los autores no se responsabilizan de posibles cambios que pudiesen realizarse en los temas dictados en la misma, así como tampoco de errores involuntarios que pudiesen existir en dicho resumen.

Por favor, mejorá este documento en github
https://github.com/ResumenesFaMAF/resumenIngenieria1

1 Introducción

Software: Colección de programas, procedimientos, y la documentación y datos asociados que determinan la operación de un sistema de computación.
Dominio del problema:

	Alumno	Industria
Error (bug)	Tolerable	No Tolerable
Interfaz	No importante	Muy importante
documentación	No existe	Existe: Usuario y proyecto
Confiabilidad y robustez	No importante	Fundamental
Inversión	No Existe	Fuerte
Portabilidad	No importante	Clave

En software las *fallas* NO son consecuencia del uso y el deterioro. Las fallas ocurren como consecuencia de errores introducidos durante el desarrollo.
Mantenimiento:

- **Correctivo** (updates) errores.
- **Adaptativo** (upgrade) funcionalidad.

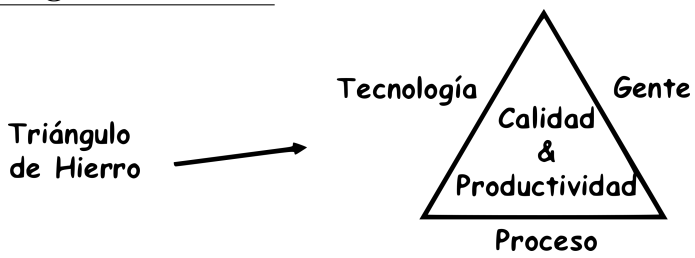
Ingeniería de Software: Aplicación de un enfoque sistemático, disciplinado, y cuantificable al desarrollo, operación, y mantenimiento del software.
Enfoque sistemático: Metodología y prácticas existentes para solucionar un problema dentro de un dominio determinado. Esto permite repetir el proceso y da la posibilidad de predecirlo (independientemente del grupo de personas que lo lleva a cabo).
Factor principal: Satisfacer necesidades cliente/usuario

Factores de impacto:

- **Escala:** Debe funcionar para entradas pequeñas y grandes.
- **Productividad:** Reducir las *KLOC/PM*.
- **Calidad:** Densidad de defectos (*#defectos / tamaño*) (+ fallas ⇒ - confiable)
  - **Funcionalidad** Capacidad de proveer funciones que cumplen las necesidades establecidas o implicadas.
  - **Confiabilidad** Capacidad de realizar las funciones requeridas bajo las condiciones establecidas durante un tiempo específico.
  - **Usabilidad** Capacidad de ser comprendido, aprendido y usado.
  - **Eficiencia** Capacidad de proveer desempeño apropiado relativo a la cantidad de recursos usados.
  - **Mantenibilidad** Capacidad de ser modificado con el propósito de corregir, mejorar, o adaptar.
  - **Portabilidad** Capacidad de ser adaptado a distintos entornos sin aplicar otras acciones que las provistas a este propósito en el producto.

- **Consistencia y repetitividad:** Sucesiva producción de sistemas de alta calidad y con alta productividad que pueda ser repetido.
- **Cambio:** Adaptarse a las necesidades.

Triángulo de hierro:

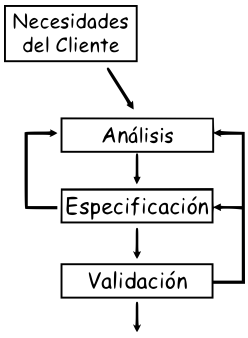


Fases del proceso de desarrollo:

- **Análisis de requisitos y especificación**
- **Arquitectura y Diseño**
- **Codificación**
- **Testing**
- **Entrega e instalación**

2 Requerimientos de Software

La *SRS* **especifica** lo que el sistema propuesto debe hacer.
Establece bases de un acuerdo entre el *cliente / usuario* y el desarrollador.
Pasos para crear una *SRS*:



1. **Análisis del problema o requerimientos.**
2. **Especificación de los requerimientos.**
3. **Validación.**

2.1 Análisis del problema o requerimientos.

Objetivo: Lograr una buena comprensión de las necesidades, requerimientos y restricciones del software. Trata con el dominio del problema. Se utilizan técnicas de diagramas de flujo de datos (*DFD*), diagramas de objetos, etc.
Principio básico: *Divide y conquistarás* para comprender cada subproblema y la relación entre ellos con respecto a:

- **Funciones** (Análisis estructural)

- **Objetos** (Análisis objetos)

- **Eventos del sistema** (particionado de eventos)

2.1.1 Modelado de flujo de datos

Un *DFD* es una representación del flujo de datos a través del sistema.

- Ve al sistema como una trandormación de *I / O*
- La transformación se realiza a través de "*Transformadores*"
- Captura la manera en que ocurre la transformación de la entrada en la salida a medida que los datos se mueven a través de los transformadores.
- No se limita al software.

2.1.2 Modelado orientado a objetos

Un sistema es visto como un conjunto de objetos interactuando entre sí (o con el usuario) a través de servicios que cada uno provee.
Objetivo:

- Identificar los objetos en el dominio del problema
- Definir las clases identificando cual es la información del estado que esta encapsula
- Identificar las relaciones entre los objetos de las distintas clases, ya sea en la jerarquía o a través de llamadas a métodos.

2.1.3 Prototipado

Clientes, usuarios y desarrolladores lo utilizan para comprender mejor el problema y las necesidades. Dos enfoques:

1. **Descartable:** (*más adecuado*) El prototipo se construye con la idea de desecharlo luego de culminada la fase de requerimientos.
2. **Evolucionario:** Se contruye con la idea de que evolucionará al sistema final.

2.2 Especificación de requerimientos:

2.2.1 Características de una SRS

- **Correcta:** Cada requerimiento representa precisamente alguna característica deseada en el sistema final.
- **Completa:** Todas las características deseadas están descritas.
- **No ambigüa** Cada requerimiento tiene exactamente un significado.
- **Consistente** Ningún requerimiento contradice a otro.
- **Verificable** Cada requerimiento se debe poder verificar.
- **Rastreable** Se debe poder determinar el origen de cada requerimiento y cómo éste se relaciona a los elementos del software.
  Requerimiento $\Rightarrow$ parte de código, Parte de código $\Rightarrow$ Requerimiento.
- **Modificable** Incorporar cambios fácilmente preservando completitud y consistencia. No redundancia.
- **Ordenada en aspectos de importancia y estabilidad** Orden de prioridades para reducir riesgos debido a cambios de requerimientos.

2.2.2 Una SRS debe especificar requerimientos sobre

1. **Funcionalidad:** Especifica toda la funcionalidad que el sistema debe proveer, las salidas que debe producir para cada entrada y las relaciones entre ellas, todas las operaciones que el sistema debe realizar. Las entradas válidas y el comportamiento del sistema para entradas inválidas, errores u otras situaciones anormales.
2. **Requerimientos de desempeño:** Todas las restricciones en el desempeño del sistema de software.

   - **Requerimientos dinámicos:** Especifican restricciones sobre la ejecución.
   - **Requerimientos estáticos:** No imponen restricción en la ejecución.

   Todos los requisitos se especifican en términos medibles, verificables.
3. **Restricciones de diseño:** Existen factores en el entorno del cliente que pueden restringir las elecciones de diseño. Ej: Ajustarse a estándares.
4. **Requerimientos de interfaces externas:** Todas las interacciones del software con gente, harware y otros softwares deben especificarse claramente. La interfaz con el usuario debe recibir la atención adecuada.

2.2.3 Casos de uso

Conceptos básicos:

- **Actor:** Persona o sistema que interactua con el sistema propuesta para alcanzar un objetivo.
- **Actor primario:** Actor principal que inicia el caso de uso.
- **Escenario:** Conjunto de acciones realizadas con el fin de alcanzar un objetivo bajo determinadas condiciones.
- **Escenario exitoso principal:** Cuando todo funciona normalmente y se alcanza el objetivo.
- **Escenario alternativos (de extensión/ de excepción):** Cuando algo sale mal y el objetivo no puede ser alcanzado.

2.3 Validación

2.3.1 Errores más comunes

- Omisión
- Inconsistencia
- Hechos incorrectos
- Ambigüedad

La *SRS* la revisan un grupo de personas, conformado por: Autor, cliente, representantes de usuarios y de desarrolladores.

2.4 Métricas

Para poder estimar costos y tiempos y planear el proyecto se necesita "medir" el esfuerzo que demandará.

2.4.1 Punto función

Es una métrica como las *LOC*, se determina solo cno la *SRS*, define el tamaño en términos de funcionalidad

Tipos de Funciones:

- Entradas externas
- Salidas externas
- Archivos de interfaz externa
- Archivos lógicos internos
- Transacciones externas

Punto función no ajustado (UFP): $\sum_{i=1}^{5} \sum_{j=1}^{3} W_{ij} C_{ij}$

Características:
1. Comunicación de datos
2. Procesamiento distribuido
3. Objetivos de desempeño
4. Carga en la configuración de operación
5. Tasa de transacción
6. Ingreso de datos online
7. Eficiencia del usuario final
8. Actualización online
9. Complejidad del procesamiento lógico
10. Reusabilidad
11. Facilidad para la instalación
12. Facilidad para la operación
13. Múltiples sitios
14. Intención de facilitar cambios

Factor de ajuste de complejidad (CAF): $0.65 + 0.01 \sum_{i=1}^{14} Pi$

Puntos función: CAF * UFP

2.4.2 Métrica de calidad

- Directa: Evalúan la calidad del documento estimando el valor de los atributos de calidad de la SRS.

- Indirecta: Evalúan la efectividad de las métricas del control de calidad usadas en el proceso en la fase de requerimientos.

3 Arquitectura del software

Definición: Es la estructura del sistema que comprende los elementos del software, las propiedades externamente visibles de tales elementos y la relación entre ellos.

- Diseño de mas alto nivel.
- Elecciones de tecnologias, productos a usar, servidores.
- Empezar a evaluar confiabilidad y desempeño.
- Divide al sistema en partes lógicas independientes.

3.1 El rol de la arquitectura

3.1.1 Comprensión y comunicación

- Al mostrar la estructura de alto nivel del sistema, la descripción arquitectónica facilita la comunicación.

- Define un marco de comprensión común entre los distintos interesados (usuarios, cliente, arquitecto, diseñador, etc.).

3.1.2 Reuso

- Principal técnica para incrementar la productividad.
- Una forma de reuso es componer el sistema con partes existenes, reusadas, otras nuevas.
- La arquitectura es muy importante: se elige una arquitectura tal que las componentes existentes encajen adecuadamente con otras componentes a desarrollar.
- Las decisiones sobre el uso de componentes existentes se toman en el momento de diseñar la arquitectura.

3.1.3 Construcción y evolución

- La división provista por la arquitectura servirá para guiar el desarrollo del sistema.
- Decide las partes a cambiar para incorporarse nuevas características en la evolución del software.

3.1.4 Análisis

- Permite considerar distintas alternativas de diseño hasta encontrar los niveles de satisfacción deseados.

3.2 Vistas de la arquitectura

3.2.1 Módulos

- Un sistema es una colección de unidades de código

- La relación entre ellos esta basada en el código.

3.2.2 Componentes y conectores

- Los elementos son entidades de ejecucción (componentes).

- Los conectores proveen el medio de interacción entre los componentes.

3.2.3 Asignación de recursos

- Como las unidades de *SW* se asignan a *SW*.

- Exponen propiedades estructurales. Ej: que archivo reside donde.

3.3 La vista componentes y conectores

Componentes: Elementos computacionales o de almacenamiento de datos.

Conectores: Mecanismos de interacción entre las componentes.

La vista de C&C describe una estructura en ejecucción del sistema: que componentes existen y como interactúan entre ellos en tiempo de ejecucción.

3.4 Estilos arquitectónicos para la vista C&C

- Un estilo arquitectónico define una gamilia de arquitecturas que satisface las restricciones de ese estilo.
- Los estilos proveen ideas para crear arquitecturas de sistemas.
- Distintos estilos pueden conbinarse para definir una nueva arquitectura.

3.5 Tubos y filtros

- Adecuado para sistemas que realizan transformación de datos
- Un solo tipo de componente: **FILTRO**
- Un solo tipo de conector: **TUBO**

3.5.1 Filtro

- Realiza transformaciones y le pasa los datos a otro filtro.
- Entidad independiente y asincrona.
- No necesita saber la identidad de los filtros a los que envía y recibe datos.
- Se encarga de hacer “buffering”.

3.5.2 Tubo

- Canal unidireccional que transporta un flujo de datos de un filtro a otro
- Solo conecta dos componentes

3.6 Datos compartidos

3.6.1 Componentes

- Repositorio de datos: Provee almacenamiento permanente confiable.
- Usuarios de datos: Acceden a los datos en el repositorio, realizan cálculos y ponen los resultados otra vez en el repositorio. La comunicación entre los usuarios de los datos sólo se hace a través del repositorio.

3.6.2 Conector

- Lectura / escritura

3.6.3 Variantes

- Estilo pizarra: Cuando se agregan/modifican datos en el repositorio se informa a todos los usuarios.
- Estilo repositorio: El repositorio es pasivo.

3.7 Estilo cliente-servidor

3.7.1 Componentes

- 2 tipos: Clientes y servidores
- Los clientes sólo se comunican con el servidor, y no con otros clientes.
- La comunicación es iniciada por el cliente, envia una solicitud al servidor y espera una respuesta.
- Comunicación usualmente sincronica.
- Usualmente cliente y servidor en distintas máquinas.

3.7.2 Conector

- Solicitud / respuesta, es asimétrico

3.8 Estilo publicar-suscribir

3.8.1 Componentes

- Dos tipos de componentes: las que publican eventos y las que se suscriben a eventos. Cada vez que un evento es publicado se invoca a las componentes suscriptas a dicho evento.

3.9 Estilo peer-to-peer

- Un único tipo de componente.
- Cada componente le puede pedir servicios a otro modelo de computación orientado a objetos.

3.10 Estilo de procesos que se comunican

- Procesos que se comunican entre sí a través de pasaje de mensajes.

3.11 Relacion/diferencia entre diseño y arquitectura

- La arquitectura es un diseño: se encuentra en el dominio de la solución y no en el dominio del problema.

- La arquitectura es un diseño de muy alto nivel que se enfoca en las componentes principales.

- Lo que usualmente llamamos diseño se enfoca en los módulos que finalmente se transformarán en el código de tales componentes.

3.12 Preservación de la intergridad de la arquitectura

- La arquitectura impone restricciones que deben preservarse en la implementación, inclusive.

- Para que la arquitectura tenga sentido, ésta debe acompañar el diseño y el desarrollo del sistema.

3.13 Método de análisis ATAM

Analiza las propiedades y las conexiones entre ellas

1. Recolectar escenarios
    - Elegir los escenarios de intereces para el analisis
    - incluir escenarios exxcepcionales solo si son importantes
2. Recolectar requerimientos y/o restricciones
    - Definir lo que se espera del sistema en tales escenarios.
    - Deben especificar los niveles deseados para los atributos de interés (preferiblemente cuantificados).
3. Describir las vistas arquitectónicas
    - Las vistas del sistema que serán evaluadas son recolectadas.
    - Distintas vistas pueden ser necesarias para distintos análisis.
4. Análisis especificos a cada atributo
    - Se analizan las vistas bajo distintos escenarios separadamente para cada atributo de interés distinto.
    - Determina los niveles que la arquitectura puede proveer en cada atributo.
    - Se comparan con los requeridos.
    - Esto forma la base para la elección entre una arquitectura u otra o la modificación de la arquitectura propuesta.
5. Identificar puntos sensitivos y de compromisos
    - Análisis de sensibilidad: cuál es el impacto que tiene un elemento sobre un atributo de calidad. Los elementos de mayor impacto son los puntos de sensibilidad.
    - Análisis de compromiso: Los puntos de compromiso son los elementos que son puntos de sensibilidad para varios atributos.