Actividad n° 1:
Atributos de calidad / Requisitos no funcionales

---

1. **¿Qué es la calidad de software?**
   R: La calidad de software hace referencia a todos los conceptos que guardan relación con la apreciación de un software en distintos ámbitos, tales como usabilidad, seguridad, mantenibilidad y todos los requisitos funcionales o “ilities” que tienen efecto en la calidad de un software
2. **¿Cuáles son los atributos de calidad de un software?**
   R: Aunque los atributos de calidad guardan estrecha relación con el tipo del software,existen cualidades o atributos que son valoradas en general , estas son :
   - Mantenibilidad
   - Eficiencia
   - Usabilidad
   - Confiabilidad
3. **¿Qué es un requisito no funcional? ¿Cuál es la diferencia con un requisito funcional?**
   R: Como se mencionó anteriormente un requisito funcional son cualidades del software que tienen impacto en la calidad del mismo, es una evaluación de cómo realiza un requisito funcional, siendo este último, las “acciones” o funciones de nuestro software.
4. **¿Es deseable cumplir con todos los atributos de calidad? ¿Por qué?**
   ¿Es deseable? Sí, pero en la práctica es difícil de reproducir, ya que para abarcar todos los atributos de calidad se requiere de tiempo y un manejo en todo momento del desarrollo del software.
5. **¿Cuál es el rol del arquitecto de software?**
   R: El rol del arquitecto de software, como bien se puede inferir de su nombre , consiste en diseñar y gestionar el desarrollo de un software, teniendo en cuenta cumplir con aspectos de calidad, así como los requisitos solicitados por el cliente.[1

Estilos arquitectonicos

---

1. **¿Qué es un estilo arquitectónico?**
   R:[1]Un «estilo» de arquitectura es un conjunto de decisiones de diseño arquitectural
que son aplicables en un contexto de desarrollo específico, restringen las decisiones de diseño de un sistema a ese contexto y plantean como objetivo ciertas cualidades para el sistema resultante.

Establecen un vocabulario común, donde se dan nombres a los componentes y conectores así como a los elementos de datos Establecen un conjunto de reglas de configuración, como la topología del sistema Definen una semántica para las reglas de composición de los elementos Posibilitan el análisis de los sistemas construidos sobre el estilo Algunos ejemplos de estilos arquitecturales:

Influenciados por los Lenguajes de Programación

- Programación estructurada
- Orientado a Objetos
- Capas
- Máquinas Virtuales
- Cliente Servidor
- n-Tier

2. **¿Cuáles son los principales estilos arquitectónicos?**
   R:[2] En un estudio comparativo de los estilos, Mary Shaw [Shaw94] considera los siguientes, mezclando referencias a las mismas entidades a veces en términos de "arquitecturas", otras invocando "modelos de diseño":
   - Arquitecturas orientadas a objeto 10
   - Arquitecturas basadas en estados
   - Arquitecturas de flujo de datos
   - Arquitecturas de control de realimentación
   - Arquitecturas de tiempo real
   - Modelo de diseño de descomposición funcional
   - Modelo de diseño orientado por eventos
   - Modelo de diseño de control de procesos
   - Modelo de diseño de tabla de decisión
   - Modelo de diseño de estructura de datos

Referencias:
[1]http://blog.osoft.es/index.php/2016/03/24/estilos-y-patrones-de-arquitecturas-de-software/
[2]http://carlosreynoso.com.ar/archivos/arquitectura/Estilos.PDF

OAuth2

---

1. **¿Qué es?**
   R:
   - Es un estándar abierto para la autorización de APIs, que nos permite compartir información entre sitios sin tener que compartir la identidad.[2]
   - OAuth 2 es un framework de autorización, que permite a las aplicaciones obtener acceso (limitado) a las cuentas de usuario de determinados servicios, como Facebook, GitHub, Twitter, Steam, BitBucket, LinkedIn y muchos más.[3]
   - OAuth 2 provee un flujo de autorización para aplicaciones web, aplicaciones móviles e incluso programas de escritorio.[3]

2. **¿Quién lo utiliza?**
   R: Es un mecanismo utilizado a día de hoy por grandes compañías como Google, Facebook, Microsoft, Twitter, GitHub o LinkedIn, entre otras muchas.[2]

3. **¿Por qué se utiliza?**
   R:
   - Surge para paliar la necesidad que se establece del envío continuo de credenciales entre cliente y servidor [2]
   - Con OAuth2 el usuario delega la capacidad de realizar ciertas acciones, no todas, a las cuales da su consentimiento para hacerlas en su nombre [3]

4. **¿Cuáles son sus limitaciones?**
   R:El principal inconveniente es que este estándar no es infalible, por lo que algunos actores de amenazas podrían aprovecharse de algún usuario para obtener amplios accesos a sus cuentas en línea e incluso robar sus credenciales de acceso, exponiéndose a otro tipo de ataques.
   Cover Redirect Vulnerability es una redirección no validada. Esta vulnerabilidad consiste en que ciertos sitios web, que implementan OAuth 2.0 como mecanismo de autenticación,aceptan parámetros dentro de la URL que provocan una redirección de los usuarios a otros sitios web sin la correcta validación de los mismos, incluso a dominios diferentes. El problema está en que en esta redirección se incluyen datos tan sensibles como el token o código de acceso que utiliza OAuth 2.0 para acceder a la información de los usuarios, por lo que no solamente presenta implicaciones sobre la seguridad, sino que también en el cumplimiento de la LOPD (LEY DE PROTECCIÓN DE DATOS).

Referencias:
[1] https://sg.com.mx/revista/33/el-rol-del-arquitecto-software
[2] https://openwebinars.net/blog/que-es-oauth2/
[3] https://programacionymas.com/blog/protocolo-oauth-2

Actividad n° 2:

1.-¿Qué son los principios SOLID?[1][2]
R[1][2]: SRP: *Principio de Responsabilidad Única*- Responsabilidad única que debería tener cada programa con una tarea bien específica y acotada.

OCP: *Principio Abierto-Cerrado*-Toda clase, modulo, método,etc. debería estar abierto para extenderse pero debe estar cerrado para modificarse.

LSP:*Principio de Substitución de Liskov*- Si la clase A es de un subtipo de la clase B, entonces deberíamos poder reemplazar B con A sin afectar el comportamiento de nuestro programa.

ISP: *Principio de Segregación de Interfaces*- Ningún cliente debería estar obligado a depender de los métodos que no utiliza.

DSI: *Principio Inversión de Dependencia*- No deben existir dependencias entre los módulos, en especial entre módulos de bajo nivel y alto nivel.

2.-¿Cuáles son los argumentos en favor y en contra de SOLID
R[3]:

- **A favor:**
  - Cambios en partes del programa no afectan otras no relacionadas
  - No se necesitan implementar métodos innecesarios.
- **En contra:**
  - Muchos módulos pequeños
  - Se crean muchas interfaces
  - Generan retrasos en la etapa de implementación

3.-¿A qué requisitos no funcionales responden? ¿A qué requisitos no funcionales no responden?

- Responden a : Modularidad,adaptabilidad, robustez
- No responden a o no están orientados a : Usabilidad, safety, security

Referencias:

[1] https://www.pinterest.com.mx/pin/480900066464005289/
[2]https://es.slideshare.net/smrocco/principios-solid
[3]https://es.slideshare.net/iwish1hadanose/do-we-need-solid-principles-during-software-development