ARQUITECTURA DE SOFTWARE

UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3

DOCENTE
Robinson Coronado

Bryan Zuleta
Weimar Quintero

Facultad de Ingeniería
Ingeniería de Sistemas
Universidad de Antioquia
2019-2

REQUISITOS NO FUNCIONALES

Los requisitos no funcionales se pueden definir como las propiedades o cualidades que el producto debe tener. Estos imponen restricciones generalmente en la etapa de diseño e implementación.

1. **Performance:** El requisito no funcional de performance se define como el tiempo máximo que puede llevar una actividad (tiempo de respuesta) y se toma como un requerimiento de calidad. Este describe el nivel de calidad del servicio y especifica propiedades relevantes que debe reunir el sistema en términos de características del estándar de calidad internacional, tales como la usabilidad, performance y seguridad; son aspectos críticos para el sistema. Estos, incluso, se identifican durante el relevamiento y el diseño del sistema. Este requisito se debe establecer para cada caso de uso en particular en la etapa de diseño.

2. Scalability: Se puede definir de la siguiente manera: “Qué tan bien puede trabajar la solución al problema cuando el tamaño de ese problema se incremente.” Esto nos indica que alguna característica del problema puede crecer eventualmente, pero un problema puede tener varios aspectos tales como conexiones simultáneas, tamaño de los datos y distribución. Éste requisito se debe tener en cuenta porque los usuarios examinaran con detenimiento hasta que punto el proyecto sobrepasará sus expectativas(tiempos de respuesta, cantidad de usuarios, recuperación ante fallos) y se adaptara a los cambios abruptos. Este requerimiento está muy asociado a la calidad y se define a partir de la etapa de Testing.

3. Volume or Volume Testing: Se define como una prueba de volumen de software, donde el software está sujeto a un gran volumen de datos. También se le conoce como prueba de inundación. La prueba de volumen se realiza en la etapa de Testing para analizar el rendimiento del sistema aumentando el volumen de datos en la BD. Con la ayuda de esto se puede estudiar el impacto en el tiempo en la respuesta y

comportamiento del sistema cuando se expone a un gran volumen de datos. Un ejemplo es probar el comportamiento cuando muchos usuarios descargan una misma canción.

4. Availability: Está relacionado con la confiabilidad de una aplicación y se concibe desde la etapa de testing. Si una aplicación no está disponible para su uso cuando se necesita, entonces es improbable que cumpla sus requerimientos. Es fácil de especificar y medir. Muchas aplicaciones deben estar disponibles todo el tiempo. Una buena medida sería el porcentaje de tiempo disponible que sería “tiempo disponible/tiempo requerido”. Las fallas disminuyen la disponibilidad y confiabilidad de una aplicación. Otra medida para esto es el tiempo medido entre fallas. La duración del sistema fuera de servicio depende también del tiempo que se demore en estar reparado.

5. Reliability: Se define como la capacidad de la aplicación para mantener su nivel de ejecución cuando esta se usa bajo las condiciones especificadas.Tal y como explicamos esta se relaciona con la disponibilidad, porque ambas se afectan entre sí. Si hay poca disponibilidad entonces hay poca confiabilidad. Por eso se debe mantener una medida entre ambas para que pueda funcionar como se debe la aplicación. Se define desde la etapa de testing.

6. Recoverability: El requisito de recuperación es una actividad de testing que consiste en probar que tan bien puede recuperarse una aplicación ante fallas de hardware y otros problemas similares. Es la falla forzada del software en una variedad de formas para verificar que la recuperación se realice correctamente. Las pruebas de recuperación no se deben confundir con las pruebas de confiabilidad, estas intentan descubrir el punto específico de la falla, las de recuperación se realizan para verificar qué tan rápido y mejor puede recuperarse la aplicación frente a cualquier tipo de bloqueo o falla. Para ello se simulan pruebas de falla o se causan fallos en un entorno controlado. Después de una falla, el mecanismo se prueba para garantizar que los datos no se

pierdan o corrompan y que se mantengan los niveles de servicios acordados.

7. Maintainability: Aquí se busca medir la facilidad con la que se puede dar mantenimiento al software. Hay varias formas de hacer un mantenimiento:

- corregir defectos
- reparar o reemplazar componentes defectuosos o desgastados sin necesidad de reemplazar piezas que todavía funcionan
- prevenir condiciones de trabajo inesperadas
- Maximizar la vida útil de un producto
- Maximizar su eficiencia, su seguridad y su fiabilidad para facilitar mantenimientos futuros o hacerlo frente a un entorno cambiado.

En algunos casos, la mantenibilidad implica un sistema de mejora continua: aprender del pasado para mejorar la capacidad de mantener sistemas o mejorar la confiabilidad de los sistemas en función de la experiencia de mantenimiento. Es un requisito que es concebido desde que inicia el ciclo de vida del software.

8. Serviceability: Se refiere a la capacidad del personal técnico para instalar, configurar y monitorear productos informáticos, identificar excepciones o fallos, depurar o aislar fallas en el análisis y proporcionar mantenimiento de hardware o software para resolver un problema y restaurar el producto en servicio. La incorporación de funciones de facilitación de servicio generalmente resulta en un mantenimiento más eficiente del producto y reduce los costos operativos y mantiene la seguridad del negocio. Este requisito se asocia más con la calidad del servicio y se pone en marcha una vez inicia el funcionamiento del software.

9. Security: Por lo general se asocia con aplicaciones que manejan datos personales o información bancaria. Se concibe desde la etapa de diseño. Hay varios requerimientos de seguridad, en los cuales están:

- **Autenticación:** Que es la capacidad de verificar la identidad de los usuarios y otras aplicaciones con las que se comunique el software.
- **Autorización:** Los derechos de los usuarios, los cuales podrían tener accesos de lectura o escritura en algunas porciones de los datos ó la posibilidad de realizar algunas operaciones, la idea es que no todos los usuarios tienen los mismos derechos, como mínimo, el usuario administrador tiene derechos diferentes del usuario cliente.
- **Encriptamiento:** la capacidad de hacer ilegibles para terceros los mensajes enviados hacia/desde la aplicación
- **Integridad:** Asegurar que el contenido de los mensajes no sean alterados durante su viaje por red.

10. Escrow: El fideicomiso o licencia es generalmente solicitado por un licenciatario para garantizar el mantenimiento del software en lugar de abandonarlo. El código fuente del software se entrega al licenciatario, si el licenciante se declara en bancarrota o no logra mantener y actualizar el software entonces este pierde la licencia.
Como la operación y mantenimiento del software son críticos para muchas compañías, por lo general desean asegurarse de que continúe, incluso si el licenciante no puede hacerlo, como por bancarrota. Esto se logra fácilmente al obtener una copia del código fuente actualizado. Sin embargo el licenciante no estará a menudo dispuesto a aceptar esto, ya que el código fuente generalmente representa uno de sus secretos comerciales mejor guardados. Como solución a este conflicto de intereses, la custodia del código fuente garantiza que el licenciatario obtenga acceso al código fuente solo cuando no pueda garantizar el mantenimiento del software, tal como se define en las condiciones acordadas. Este requerimiento se contempla desde antes de la etapa de diseño del proyecto, ya que tiene más relación con la parte legal del contrato entre las partes interesadas.

WEBGRAFÍA

- https://pdfs.semanticscholar.org/2188/b5c66839fefdf4e513993acbe9524d9d7f79.pdf
- http://utpingsof1.blogspot.com/2008/11/los-requerimientos-no-funcionales.html
- https://www.guru99.com/volume-testing.html