Availability, Interoperability, modifiability, Performance, Security, Testability, Usability, others

Pag 137

Modifiability atributo de calidad

Modificabilidad o facilidad para modificar el código.

Consideraciones

Los cambios en el código son parte de los sistemas y representa el costo más alto durante el ciclo de vida de un sistema de software.

1 que puede cambiar ?

El código puede tener la necesidad de cambio en cualquier aspecto como plataforma (software, hardware, protocolos, calidades del sistema, capacidad, etc.)

2 cuál es la tendencia de los cambios ?

Es casi imposible determinar la tendencia de los cambios, pero el arquitecto tiene que tomar la decisión de cuál cree que pueda ser la tendencia y considerar que decisiones pueden ser tomadas de antemano, y visualizar cuales cambios podrán ser soportados y cuáles no.

3 donde se hará el cambio y quien lo efectuará ?

Lo mas común es que los cambios los realiza el programador pero por las herramientas y circunstancias actualmente los cambios pueden ser en:

En la implementación (modificando el código fuente).

Durante la compilación (usando parámetros para el tiempo de compilación)

Durante la construcción de ejecutables (con la selección de librerías)

Durante la configuración (con la selección de parámetros de instalación, plugins, etc.)

4 cuál es el costo del cambio ?

Hacer un sistema mas fácil de modificar implica 2 tipos de costos

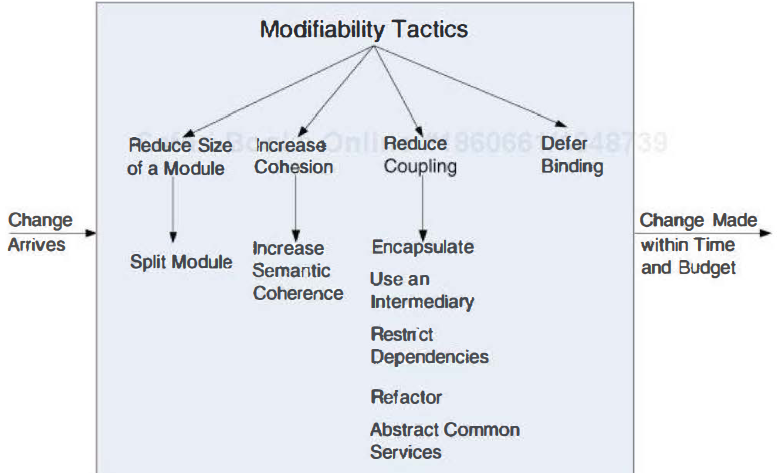
El costo de introducir el mecanismo para hacer mas sencillo de modificar (ejemplo un archivo de parámetros o variables.)

El costo de hacer la modificación usando el mecanismo

Escenario

|  |  |
| --- | --- |
| Fuente de estimulo | Se especifica quien hace el cambio: el programador, el desarrollador, el administrador del sistema. |
| Estimulo | El cambio a hacerse que puede ser:  agregar una función,  modificar una función,  eliminar una función  cambio a las cualidades del sistema:  hacerlo más responsivo  aumentar su disponibilidad  la capacidad  agregar nueva tecnología |
| Artefactos | Componentes o módulos específicos. |
| Entorno | Especificación del área donde se realizará el cambio  Tiempo de diseño  Tiempo de compilación  Tiempo de construcción  Tiempo de ejecucion |
| Respuesta | Hacer el cambio,  probarlo el cambio,  integrar el cambio al sistema real |
| Medición de la respuesta | Todas las posibles respuestas involucran tiempo y dinero entonces estas son las medidas más comunes:  -Usar herramientas para contar los tiempos que llevan las modificaciones, incluir las nuevas modificaciones generadas a partir de la modificación original.  -Hacer una conversión del tiempo a dinero, depende del salario o salarios de los empleados involucrados en la ejecución de los cambios.  Otras métricas: Líneas de código, complejidad del código, esfuerzo, nuevos defectos introducidos. |

Tácticas



1 reducción de tamaño de los módulos

1.1 división de modulo

2 Incrementar Cohesión

Hay varias tácticas que implican mover responsabilidades de un módulo a otro con la intención de reducir los efectos laterales de los cambios.

2.1 Incrementar la coherencia semántica

Si las responsabilidades A y B de un módulo no sirven al mismo propósito entonces deberán estar en módulos diferentes

3 Reduce coupling

3.1 Encapsulate

Introduce una interface explicita a un módulo, la interface incluye la API, es la táctica mas

común y reduce la probabilidad de que un cambio se propague a otro modulo.

Las interfaces se diseñan para aumentar la modificabilidad.

3.2 Use an intermediary

Dada una dependencia entre A y B, la dependencia puede romperse usando un intermediario,

el tipo de intermediario depende del tipo de dependencia.

3.3 Restrict dependencies

Restringe los modulos que puedan interactuar o depender de otro modulo, esta táctica se usa

en arquitecturas en capas en donde una capa solo es usada con capas inferiores.

3.4 Refactor

Es la eliminación de que un cambio afecte a 2 o más módulos, la eliminación más común es la

de responsabilidades en común.

3.5 Abstract common services

En caso de 2 o más módulos que proveen casi el mismo servicio, la implementación más

sencilla es la de implementar los servicios de modo más general (abstracta).

Un modo muy común de abstracción es parametrizar la descripción e implementación de las

actividades de un módulo.

4 Defer Binding

Es más barato prever una modificación de código haciéndolo con código a que lo haga algún programador, y el mejor modo de introducir flexibilidad son funciones parametrizadas.

Cuando por necesidad se cambian los valores en las funciones en etapas posteriores se le llama táctica *defer binding*.

Instalar un mecanismo de modo que alguien pueda hacer cambios al sistema sin modificar código sino modificar parámetros (variables), este mecanismo se le conoce como *externalizar el cambio*.

De la lista de tácticas seleccionadas contrastar contra ……

**Checklist de diseño (contrastar con Categorías de decisiones de diseño arquitecturales)**

**Son para contrastar si las decisiones tomadas cumplen esas preguntas o sentencias (asignaciones)**

|  |  |
| --- | --- |
| Asignación de responsabilidades | Determinar cuál de las responsabilidades del sistema necesitan operar con otros sistemas.  Asegurar:  -aceptar el request  -intercambiar la información  -rechazar el request  -notificar a las entidades involucradas (sistemas, personas)  -corregir o enmascarar la fault o failure  -Loggear el request (almacenar en bitácoras Log) |

|  |  |
| --- | --- |
| Modelo de coordinación | Asegurar que el mecanismo de coordinación satisfaga los requerimientos  Considerar:  -el volumen de tráfico generado por tu sistema y generado por otros sistemas que no controlas.  -tiempo de envió de mensajes entre sistemas  -registro de mensajes enviados por otros sistemas  -inestabilidad de los mensajes enviados por “tus” sistemas |

|  |  |
| --- | --- |
| Modelo de datos | Determinar la sintaxis y semántica de las abstracciones de datos mayores y que pueda ser utilizado para intercambiar información.  -asegurar que las abstracciones de datos mayores sean consistentes con los datos de los sistemas con los que interactúa, es posible que se hagan transformaciones desde o para las abstracciones de datos de otros sistemas con los que interactúa. |

|  |  |
| --- | --- |
| Mapeo entre elementos arquitecturales | Para la interoperabilidad el mapeo o correspondencia más crítica o complicada es la de componente a procesadores.  Aunque es necesario que los componentes estén soportados por procesadores que accedan a las redes, lo más importante es la seguridad y disponibilidad. |

|  |  |
| --- | --- |
| Administración de recursos | Asegurar que la interoperación con otros sistemas (aceptar o rechazar solicitudes) nunca llegue a un estado crítico como “cuello de botella”.  Asegurar un nivel aceptable de recurso para soportar la comunicación de las solicitudes.  Asegurar que haya un buen protocolo para la asignación de recursos. |

|  |  |
| --- | --- |
| Binding time | Determinar cómo los sistemas pueden alcanzar una interoperación y como se reconocen entre sí mismos.  Por cada sistema asegurar:  -asegurar la existencia de una política para el manejo de conexión con sistemas conocidos y desconocidos.  -asegurar que existan y funcionen los mecanismos para rechazar solicitudes y que queden registrados en bitácora (loggeados).  -en caso de conexión tardía asegurar los mecanismos que soporten el descubrimiento (rastreo) de nuevos servicios o protocolos, o el envío de información usando protocolos establecidos. |

|  |  |
| --- | --- |
| Selección de tecnología | De las tecnología seleccionadas, son visibles en el espacio de interfaces del sistema ? y si es visible que efectos de interoperabilidad va a haber ?  Los efectos son limitados, parciales o no hay efectos ?  Asegurar un nivel de efectos aceptables  Considere tecnologías que fueron diseñados para soportar interoperabilidad como servicios web, estos recursos pueden ser usados para satisfacer la interoperabilidad de “su” sistema ? |