Resumen Teorico Arquitectura de Software

Modificabilidad

Capacidad del sistema para que se le ingresen cambios en funcion del costo y riesgo que tiene la introduccion de estos cambios.

El mayor gasto de un sistema de software ocurre despues de que se realizo su liberacion inicial.

La modificalidad trata sobre el cambio y nuestro interes es sobre los costos y los riesgos de realizar ese cambio. A la hora de planear la modificalidad de un sistema el arquitecto debe hacerse las siguientes preguntas:

- Que puede cambiar?

El cambio puede ocurrir en cualquier aspecto de nuestro sistema: hardware, sistema operativo, los protocolos utilizados, la capacidad, etc

- Cual es la naturaleza del cambio?

No podemos plantearnos un sistema que este preparado para todo tipo de cambio ya que su costo seria demasiado costoso y se verian afectados otros atributos de calidad como consecuencia por lo que el arquitecto debe decidir que tipo de cambios seran soportados por el sistema y cuales no

- Cual es el cambio a realizar y quien lo hara?

Los cambios pueden ser realizados por desarrolladores, administradores o incluso por usuarios finales. A su vez estos cambios pueden realizarse en distintos momentos, ya sea durante la implementacion, en tiempo de compilacion, durante el build, en la configuracion o en tiempo de ejecucion

- Cual es el costo del cambio?

Hacer un sistema modificable implica dos tipos de costos: el costos de crear un mecanismo para que el sistema sea mas modificable y el costo que implica realizar un cambio utilizando el mecanismo antes implementado.

Comunmente para saber si la implementacion de un mecanismo para abordar el cambio vale la pena se realiza la comparacion entre:

N x Costo de Cambio sin Mecanismo contra N x Costo de Cambio con Mecanismo + Costo de implementacion del mecanismo

Donde N es la supuesta cantidad de cambios previstos.

Escenario de Modificabilidad

Fuente de estimulo: quien realiza el cambio

Estimulo: especificacion del cambio a realizar, por ejemplo puede ser una nueva funcionalidad, cambiar una existente o cambiar alguna caracteristica de la capacidad del sistema

Artefacto: que parte va a cambiar, siendo que componentes o modulos se veran afectados por el cambio

Ambiente: cuando se realiza el cambio. Design time, Build time, Configuration time, Runtime.

Respuesta: Hacer el cambio, probarlo y desplegarlo

Medida de la respuesta: Todas las respuestas insumen tiempo y dinero. Algunas medidas comunes son: cantidad, tamaño, complejidad y artefactos afectados por los cambios, esfuerzo que implica su realizacion, tiempo calendario insumido, cantidad de defectos introducidos por el cambio y el dinero incurrido en su realizacion.

Tacticas para la modificabilidad

Las tacticas de modificabilidad tiene como objetivo controlar la complejidad de realizar cambios, el tiempo que toman y el costo de realizacion.

Para que el impacto de los cambios tenga el menor impacto posible en distintos módulos del sistema y que las razones por la cual los módulos cambien sean menores debemos apuntar a tener alta cohesion y bajo acoplamiento en nuestros modulos.

Parametros de modificabilidad

* Tamaño del modulo
* Acoplamiento
* Cohesion
* Binding time of modification (momento en el que se realiza la modificación)

**Tacticas**

**Reducir tamaño de modulo**

* Dividir modulos: si modificamos un modulo que incluya varias capacidades de nuestra aplicación, la modificación del mismo sera costosa. Refinar el modulo en varios modulos mas pequeños reducirá el costo promedio en cambios futuros siempre y cuando la separación de módulos refleje el tipo de cambio que se puede llegar a realizar.

**Aumentar cohesion**

* Aumentar la coherencia semantica: Si las responsabilidades A y B de un modulo no sirven al mismo proposito, entonces deberían ser ubicadas en modulos separados. Esto puede llevar a crear un nuevo modulo o a mover la responsabilidad a un modulo existente. Un metodo para analizar si las responsabilidades deben moverse a otro modulo es suponer posibles cambios para el modulo y si ciertas responsabilidades no se ven afectadas entonces posiblemente estas deban moverse a otro modulo.

**Reducir acoplamiento**

* Encapsulamiento: Encapsulamiento inroduce una interface explicita al modulo. El encapsulamiento reduce las probabilidades de que un cambio se propague a otros modulos ya que el acoplamiento de otros modulos queda atado a la interface. La desventaja de esto es que se limita a los modulos externos a interactuar mediante los metodos provistos por la intarface. Las interfaces diseñadas para aumentar la modificabilidad deben ser abstractas con respecto a las partes del modulo que pueden llegar a cambiar.
* Uso de intermediarios: el uso de un intermediario rompe la dependencia entre las responsabilidades de un modulo A y un modulo B. Por ejemplo el intermediario publish-suscribe rompe la dependencia entre suscriptores y productores
* Restringir dependencias: consiste en restringir los modulos con los que otro modulo interactua o depende de. En la practica esto se logra modificando la visibilidad de los modulos o las autorizaciones de uso. Esta tactica se ve en arquitecturas de capas donde las capas solo pueden utilizar las capas inferiores.
* Refactor: tactica utilizada cuando dos modulos son afectados por el mismo cambio porque son duplicados uno del otro (al menos parcialmente). En estos casos la responsabilidad común se mueve para afuera de los modulos y se la situa en el lugar mas adecuado.
* Abstraer servicios comunes: para los casos en los que varios modulos implementan funcionalidades muy parecidas pero tienen servicios similares suele ser eficiente a nivel de costos implementar el servicio una única vez de forma mas abstracta. Cualquier modificación al servicio común se realizara unicamente en un lugar reduciendo el costo de modificacion.

**Defer Binding (Binding diferido)**

Como el costo del trabajo de las personas es mayor al de las computadoras, dejar que las computadores se encarguen de la mayor parte posible del cambio en la mayoría de los casos reducirá el costo de la implementación del cambio. El ejemplo mas común de un mecanismo de este tipo son los parámetros. Una función parametrizada f(a,b) es mas general que una función f(a) donde b es constante.

Tacticas para bindear en tiempo de compilación o build

* Remplazar componentes (en un build script o makefile)
* Parametrizacion en tiempo de compilación
* Aspects

Tacticas para bindear en tiempo de despliegue

* Resources files

Tacticas para bindear en tiempo de ejecucion

* Runtime registration
* Dynamic lookup
* Interpretacion de parametros
* Startup time binding
* Nombre de servidores
* Plug-ins
* Publish-suscribe
* Repositorios compartidos
* Polimorfismos

Instalar un mecanismo de modificabilidad para que alguien mas pueda hacer el cambio luego sin necesidad de modificar el código de la aplicación se llama externalizar el cambio.