MODIFICABILIDAD

Los cambios suceden para agregar nuevas funciones, para cambiar o incluso retirar las antiguas. Los cambios se producen para corregir defectos, reforzar la seguridad o mejorar el rendimiento. Se producen cambios para mejorar la experiencia del usuario. Los cambios suceden para adoptar nuevas tecnologías, nuevas plataformas, nuevos protocolos, nuevos estándares. Los cambios suceden para hacer que los sistemas funcionen juntos, incluso si nunca fueron diseñados para hacerlo.
La modificabilidad tiene que ver con el cambio, y nuestro interés en él se centra en el costo y riesgo de realizar cambios. Para planificar la modificabilidad, un arquitecto debe considerar cuatro preguntas:
■ ¿**Qué puede cambiar?** Un cambio puede ocurrir en cualquier aspecto de un sistema: las **funciones** que el sistema calcula, la **plataforma** (el hardware, el funcionamiento sistema, middleware), el **entorno** en el que opera el sistema (los sistemas con los que debe interoperar, los protocolos que utiliza para comunicarse con el resto del mundo), las **cualidades** que exhibe el sistema (su rendimiento, su fiabilidad e incluso sus futuras modificaciones), y su **capacidad** (número de usuarios admitidos, número de operaciones simultáneas).

■ **¿Cuál es la probabilidad de cambio?** No se puede planificar un sistema para todos cambios potenciales: el sistema nunca se haría, o si se hiciera sería demasiado caro y probablemente sufriría un atributo de calidad problemas en otras dimensiones. Aunque cualquier cosa puede cambiar, el arquitecto tiene que tomar decisiones difíciles sobre qué cambios son probables, y por lo tanto, qué cambios deben ser apoyados y cuáles no.

■**¿Cuándo se realiza el cambio y quién lo hace?** Más comúnmente en el pasado, se realizó un cambio en el código fuente. Es decir, un desarrollador tuvo que hacer el cambio, que se probó y luego se implementó en una nueva versión. Ahora, Sin embargo, la cuestión de cuándo se realiza un cambio está entrelazada con la cuestión de quién lo hace. Un usuario final que cambia el protector de pantalla es claramente haciendo un cambio en uno de los aspectos del sistema. Igualmente claro, es no en la misma categoría que cambiar el sistema para que pueda usarse a través de la web en lugar de en una sola máquina. Se pueden realizar cambios en el implementación (modificando el código fuente), durante la compilación (usando conmutadores en tiempo de compilación), durante la compilación (por elección de bibliotecas), durante configuración de la configuración (mediante una variedad de técnicas, incluida la configuración de parámetros),o durante la ejecución (por configuración de parámetros, complementos, etc.). Un cambio también puede ser realizado por un desarrollador, un usuario final o un administrador del sistema.

■ **¿Cuál es el costo del cambio?** Hacer un sistema más modificable implica dos tipos de costos:
■ El costo de introducir el (los) mecanismo (s) para hacer que el sistema sea más modificable
■ El costo de realizar la modificación mediante el (los) mecanismo (s)
Por ejemplo, el mecanismo más simple para realizar un cambio es esperar una solicitud de cambio para entrar, luego cambie el código fuente para acomodar el solicitud. El costo de introducir el mecanismo es cero; el costo del ejercicio

es el costo de cambiar el código fuente y revalidar el sistema. En el otro extremo del espectro es un generador de aplicaciones, como un constructor de interfaz de usuario. El constructor toma como entrada una descripción de la interfaz de usuario del diseñado producido a través de técnicas de manipulación directa y produce (generalmente) código fuente. El costo de introducir el mecanismo es el costo de construir la interfaz de usuario constructor, que puede ser sustancial. El costo de usar el mecanismo es el costo de producir el insumo para alimentar al constructor (el costo puede ser sustancial o insignificante), costo de ejecutar el constructor (aproximadamente cero), y luego el costo de lo que sea las pruebas se realizan sobre el resultado (generalmente mucho menos de lo habitual).

Para N modificaciones similares, una justificación simplificada para un mecanismo de cambio es que

N × Costo de realizar el cambio sin el mecanismo <_
Costo de instalación del mecanismo +
(N × Coste de realizar el cambio mediante el mecanismo).

N es el número anticipado de modificaciones que utilizarán la modificabilidad mecanismo, pero N es una predicción. Si entran menos cambios de los esperados, entonces es posible que no se justifique un mecanismo de modificación costoso. Además, el costo de crear el mecanismo de modificabilidad podría aplicarse en otros lugares, al agregar funcionalidad, en la mejora del rendimiento, o incluso en inversiones que no sean de software como comprar acciones tecnológicas. Además, la ecuación no tiene en cuenta el tiempo. Puede resultar más barato a largo plazo construir un mecanismo sofisticado de gestión de cambios, pero es posible que no pueda esperar a que lo haga.

7.1 Escenario general Modificabilidad

A partir de estas consideraciones, podemos ver las partes del escenario de la modificabilidad general:

■ Fuente de estímulo**.** Esta porción especifica quién hace el cambio: el desarrollador, administrador del sistema o usuario final.
■ Estímulo. Esta parte especifica el cambio a realizar. Un cambio puede ser la adición de una función, la modificación de una función existente, o la eliminación de una función. También se puede hacer un cambio en las cualidades del sistema: hacerlo más receptivo, aumentar su disponibilidad, etc.
La capacidad del sistema también puede cambiar. Acomodando un creciente el número de usuarios simultáneos es un requisito frecuente. Finalmente, cambios puede suceder que se adapte a una nueva tecnología de algún tipo, la más común de los cuales es portar el sistema a un tipo diferente de computadora o red de comunicación.
■ Artefacto. Esta parte especifica lo que se va a cambiar: componentes específicos o módulos, la plataforma del sistema, su interfaz de usuario, su entorno o otro sistema con el que interopera.
■ Medio ambiente. Esta parte especifica cuándo se puede realizar el cambio: diseño tiempo, tiempo de compilación, tiempo de construcción, tiempo de inicio o tiempo de ejecución.

■ Respuesta. Realice el cambio, pruébelo e impleméntelo.
■ Medida de respuesta. Todas las posibles respuestas toman tiempo y cuestan dinero; el tiempo y el dinero son las medidas de respuesta más comunes.
El costo generalmente significa un desembolso directo, pero también podría incluir el costo de oportunidad de que su personal trabaje en los cambios en lugar de otras tareas. Otras medidas incluyen el alcance del cambio.
(número de módulos u otros artefactos afectados) o el número de nuevos defectos introducidos por el cambio, o el efecto sobre otros atributos de calidad. Si el cambio lo está realizando un usuario, es posible que desee medir la eficacia de los mecanismos de cambio proporcionados, que en cierto modo se superponen con medidas de usabilidad
La figura 7.1 ilustra un escenario de modificabilidad concreto: el desarrollador desea cambiar la interfaz de usuario modificando el código en tiempo de diseño. Las modificaciones se realizan sin efectos secundarios dentro de las tres horas.
La Tabla 7.1 enumera los elementos del escenario general que caracterizan modificabilidad.

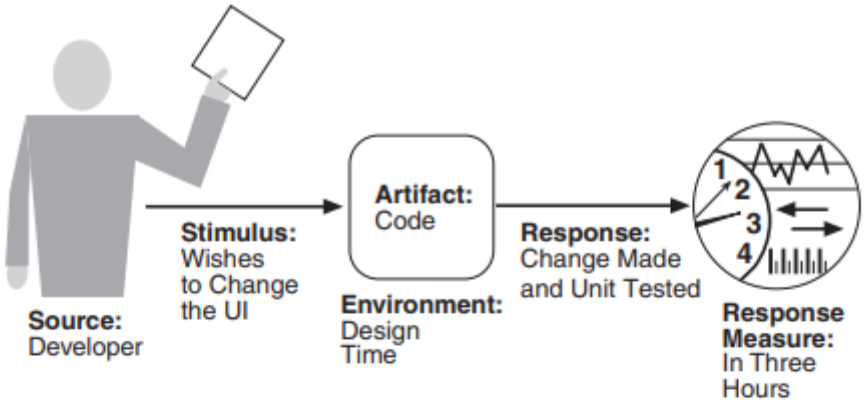

FIGURE 7.1 Sample concrete modifiability scenario

TABLE 7.1 Modifiability General Scenario

Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
Artifacts	Code, data, interfaces, components, resources, configurations, . . .
Environment	Runtime, compile time, build time, initiation time, design time
Response	One or more of the following: ▪ Make modification ▪ Test modification ▪ Deploy modification
Response Measure	Cost in terms of the following: ▪ Number, size, complexity of affected artifacts ▪ Effort ▪ Calendar time ▪ Money (direct outlay or opportunity cost) ▪ Extent to which this modification affects other functions or quality attributes ▪ New defects introduced

7.2 Tácticas para modificabilidad

Las tácticas para controlar la modificabilidad tienen como objetivo controlar la complejidad de realizar cambios, así como el tiempo y el costo para realizarlos. La figura 7.2 muestra esta relación para comprender la modificabilidad, comenzamos con el acoplamiento y la cohesión.

Los módulos tienen responsabilidades. Cuando un cambio provoca la modificación de un módulo, sus responsabilidades se modifican de alguna manera. Generalmente, un cambio que afecta un módulo es más fácil y menos costoso que si cambia más de un módulo. Sin embargo, si las responsabilidades de dos módulos se superponen de alguna manera, entonces un solo

el cambio puede afectarlos a ambos. Podemos medir esta superposición midiendo la probabilidad de que una modificación de un módulo se propague al otro. Este es llamado acoplamiento, y el acoplamiento alto es enemigo de la modificabilidad.
La cohesión mide el grado de relación entre las responsabilidades de un módulo. De manera informal, mide la "unidad de propósito" del módulo. Unidad de propósito puede medirse por los escenarios de cambio que afectan a un módulo. La cohesión de un módulo es la probabilidad de que un escenario de cambio que afecte a una responsabilidad también afectan otras responsabilidades (diferentes). Cuanto mayor es la cohesión, menor la probabilidad de que un cambio dado afecte a múltiples responsabilidades. La alta cohesión es buena; la cohesión baja es mala. La definición permite dos módulos con propósitos similares cada uno para ser cohesivo.
Dado este marco, ahora podemos identificar los parámetros que usaremos para motivar tácticas de modificabilidad:
■ **Tamaño de un módulo**. Las tácticas que dividen los módulos reducirán el costo de fabricación una modificación al módulo que se está dividiendo siempre que se elija la división para reflejar el tipo de cambio que es probable que se realice.

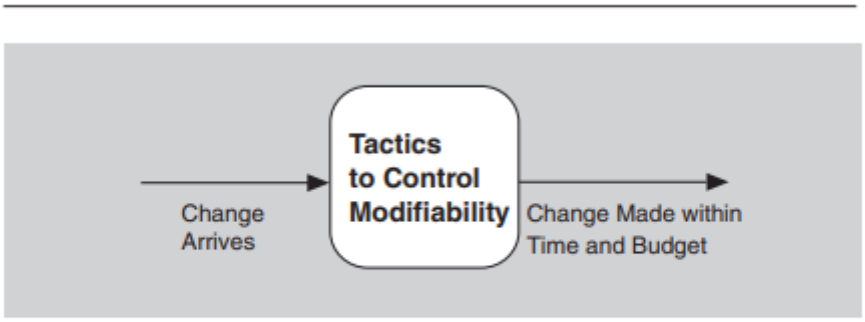

FIGURE 7.2 The goal of modifiability tactics

■ **Acoplamiento**. Reducir la fuerza del acoplamiento entre dos módulos A y B disminuirá el costo esperado de cualquier modificación que afecte a A.
Las tácticas que reducen el acoplamiento son aquellas que colocan intermediarios de varios ordena entre los módulos A y B.
■ **Cohesión**. Si el módulo A tiene una cohesión baja, entonces la cohesión se puede mejorar eliminando responsabilidades que no se vean afectadas por cambios anticipados.
Finalmente, debemos preocuparnos por cuándo en el desarrollo de software ciclo de vida se produce un cambio. Si ignoramos el costo de preparar la arquitectura para la modificación, preferimos que el cambio se consolide lo más tarde posible. Cambios solo se puede hacer con éxito (es decir, rápidamente y al menor costo) al final del ciclo de vida si la arquitectura está adecuadamente preparada para acomodarlos. Por lo tanto, El cuarto y último parámetro en un modelo de modificabilidad es este:
■ **Fecha vinculante de modificación.** Una arquitectura adecuadamente equipada para acomodar modificaciones al final del ciclo de vida costará, en promedio, menos que una arquitectura que obliga a realizar la misma modificación antes.
La preparación del sistema significa que algunos costos serán cero o muy baja, para modificaciones tardías del ciclo de vida. Esto, sin embargo, ignora el costo de preparando la arquitectura para la unión tardía.

Ahora podemos entender que las tácticas y sus consecuencias afectan a uno o más de los parámetros anteriores: reducir el tamaño de un módulo, aumentar la cohesión, reducir el acoplamiento y aplazar el tiempo de enlace. Estas tácticas se muestran en Figura 7.3

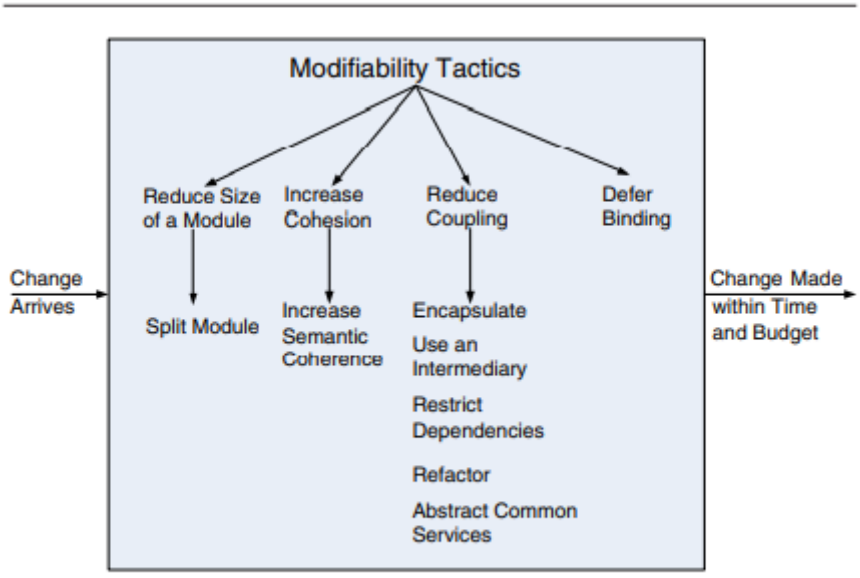

FIGURE 7.3 Modifiability tactics

Reducir el tamaño de un módulo

■ Módulo dividido. Si el módulo que se está modificando incluye una gran capacidad, es probable que los costos de modificación sean altos. Refinando el módulo en varios módulos más pequeños deberían reducir el costo promedio de cambios futuros.

Incrementar la cohesión

Varias tácticas implican trasladar responsabilidades de un módulo a otro. los El propósito de trasladar una responsabilidad de un módulo a otro es reducir la probabilidad de efectos secundarios que afecten a otras responsabilidades en el módulo original.

- Incrementar la coherencia semántica. Si las responsabilidades A y B en un módulo no tienen el mismo propósito, deben colocarse en módulos diferentes. Esto puede implicar la creación de un nuevo módulo o puede implicar trasladar una responsabilidad a un módulo existente. Un método para identificar las responsabilidades que se trasladarán es formular hipótesis sobre cambios probables que afecten a un módulo. Si algunas responsabilidades no se ven afectadas por estos cambios, entonces esas responsabilidades probablemente deberían eliminarse.

Reducir el acoplamiento

Pasamos ahora a las tácticas que reducen el acoplamiento entre módulos.

■ Encapsular. La encapsulación introduce una interfaz explícita a un módulo. Esta interfaz incluye una interfaz de programación de aplicaciones (API) y su responsabilidades asociadas, como "realizar una transformación sintáctica en un parámetro de entrada a una representación interna ". Quizás el más común

táctica de modificabilidad, la encapsulación reduce la probabilidad de que un cambio en un módulo se propaga a otros módulos. Los puntos fuertes de acoplar que anteriormente fue al módulo ahora vaya a la interfaz del módulo.
Sin embargo, estas fortalezas se reducen porque la interfaz limita las formas en el que las responsabilidades externas pueden interactuar con el módulo (tal vez a través de una envoltura). Las responsabilidades externas ahora solo pueden interactuar directamente con el módulo a través de la interfaz expuesta (interacciones indirectas, sin embargo, como la dependencia de la calidad del servicio, probablemente permanecerá sin cambios). Las interfaces diseñadas para aumentar la modificabilidad deben ser abstractas con respecto a los detalles del módulo que es probable que cambien, es decir, deberían ocultar esos detalles.

■ Utilizar un intermediario rompe una dependencia. Dada una dependencia entre la responsabilidad A y la responsabilidad B (por ejemplo, llevar a cabo A primero requiere llevando a cabo B), la dependencia se puede romper utilizando un intermediario.
El tipo de intermediario depende del tipo de dependencia.
■ Restringir dependencias es una táctica que restringe los módulos con los que un módulo determinado interactúa o de los que depende. En la práctica, esta táctica se logra restringiendo la visibilidad de un módulo (cuando los desarrolladores no pueden ver una interfaz,no puede emplearlo) y por autorización (restringiendo el acceso solo a los módulos). Esta táctica se ve en arquitecturas en capas, en las que una capa es solo permitido usar capas inferiores.
.
■ Refactorizar es una táctica que se lleva a cabo cuando dos módulos se ven afectados por el mismo cambian porque son duplicados (al menos parciales) entre sí. La refactorización de código es una práctica fundamental de los proyectos de desarrollo ágiles, como limpieza paso para asegurarse de que los equipos no hayan producido código duplicado o demasiado complejo; sin embargo, el concepto también se aplica a los elementos arquitectónicos.
Las responsabilidades comunes (y el código que las implementa) se "factorizan "fuera" de los módulos donde existen y se le asigna un hogar apropiado de los suyos. Compartiendo responsabilidades comunes, es decir, convirtiéndolas submódulos del mismo módulo principal: el arquitecto puede reducir el acoplamiento.
■ Servicios comunes abstractos. En el caso de que dos módulos brinden servicios no iguales pero similares, puede ser rentable implementar los servicios solo una vez en una forma más general (abstracta). Cualquier modificación al servicio (común) tendría que ocurrir en un solo lugar, reduciendo los costos de modificación. Una forma común de introducir una abstracción es parametrizar la descripción (e implementación) de las actividades de un módulo.
Los parámetros pueden ser tan simples como los valores de las variables clave o tan complejos como declaraciones en un lenguaje especializado que se interpretan posteriormente.

Aplazar la vinculación -Defer binding

Porque el trabajo de las personas es casi siempre más caro que el trabajo de computadoras, dejar que las computadoras manejen un cambio tanto como sea posible casi siempre reduce el costo de hacer ese cambio. Si diseñamos artefactos con una función flexibilidad, entonces ejercitar esa flexibilidad suele ser más barato que codificar manualmente un cambio específico.

Los parámetros son quizás el mecanismo más conocido para introducir flexibilidad, y eso recuerda a la táctica abstracta de servicios comunes.

La función parametrizada f (a, b) es más general que la función similar f (a) que asume b = 0. Cuando vinculamos el valor de algunos parámetros en una fase diferente en el ciclo de vida que en el que definimos los parámetros, estamos aplicando la táctica de vinculación diferida.

En general, cuanto más tarde en el ciclo de vida podamos vincular valores, mejor. Sin embargo, poner los mecanismos en su lugar para facilitar que la unión tardía tiende a ser más caro, otra compensación.

Queremos enlazar lo más tarde posible, siempre que el mecanismo que lo permita es rentable.

Las tácticas para vincular valores en tiempo de compilación o tiempo de construcción incluyen estas:

- Reemplazo de componentes (por ejemplo, en un script de compilación o archivo MAKE)
- Parametrización en tiempo de compilación
- Aspectos

Las tácticas para vincular valores en el momento del despliegue incluyen esto:

- Enlace de tiempo de configuración

Las tácticas para vincular valores en el momento de inicio o inicialización incluyen esto:

- Archivos de recursos

Las tácticas para vincular valores en tiempo de ejecución incluyen las siguientes:

- Registro en tiempo de ejecución
- Búsqueda dinámica (p. Ej., De servicios)
- Interpretar parámetros
- Vinculación de la hora de inicio
- Servidores de nombres
- Complementos
- Publicar-suscribirse
- Repositorios compartidos
- Polimorfismo

Separar la construcción de un mecanismo de modificabilidad del uso del mecanismo para realizar una modificación admite la posibilidad de diferentes partes interesadas.

Una parte interesada (generalmente un desarrollador) para proporcionar el mecanismo y otro interesado (un instalador, por ejemplo, o un usuario) para ejercitarlo posteriormente, posiblemente en una fase de ciclo de vida completamente diferente. Instalar un mecanismo para que otra persona puede realizar un cambio en el sistema sin tener que cambiar ningún código a veces se llama externalizar el cambio.