4- Entendiendo atributos de calidad

Muchos factores determinan las cualidades que deben proporcionarse en la arquitectura de un sistema.
Estas cualidades van más allá de la funcionalidad, que es la declaración básica del sistema capacidades, servicios y comportamiento. Aunque la funcionalidad y otras cualidades están estrechamente relacionados, como verá, la funcionalidad a menudo ocupa el asiento delantero en el esquema de desarrollo. Sin embargo, esta preferencia es miope. Los sistemas son frecuentemente rediseñados no porque sean funcionalmente deficientes: los reemplazos a menudo son funcionalmente idénticos, pero debido a que son difíciles de mantener, portar o escalar; o son demasiado lentos; o han sido comprometidos por piratas informáticos.
En el Capítulo 2, dijimos que la arquitectura era el primer lugar en la creación de software en qué requisitos de calidad podrían abordarse. Es el mapeo de un sistema funcionalidad en estructuras de software que determina el soporte de la arquitectura por calidades.
Un atributo de calidad (QA) es una propiedad medible o comprobable de un sistema que se utiliza para indicar qué tan bien el sistema satisface las necesidades de sus grupos de interés. Puede pensar en un atributo de calidad como una medida de la "bondad"de un producto en alguna dimensión de interés para una parte interesada.

4.1 Arquitectura y requisitos

Los requisitos para un sistema vienen en una variedad de formas: requisitos textuales, maquetas, sistemas existentes, casos de uso, historias de usuarios y más.
No importa la fuente, todos los requisitos abarcan las siguientes categorías:
1. Requisitos funcionales. Estos requisitos establecen lo que el sistema debe hacer y cómo debe comportarse o reaccionar a los estímulos en tiempo de ejecución.
2. Requisitos de atributos de calidad. Estos requisitos son calificaciones de los requisitos funcionales o del producto en general. Una calificación de un requisito funcional es un elemento como qué tan rápido debe ser la función realizado, o cuán resistente debe ser a una entrada errónea. Una calificación del producto en general es un elemento como el tiempo para implementar el producto o un limitación de los costes operativos.
3. Restricciones. Una restricción es una decisión de diseño con cero grados de libertad. Es decir, es una decisión de diseño que ya se tomó. Ejemplos incluyen el requisito de utilizar un determinado lenguaje de programación o de reutilizar un determinado módulo existente, o un mandato de gestión para hacer que su sistema esté orientado al servicio.

¿Cuál es la "respuesta" de la arquitectura a cada uno de estos tipos de requisitos?

1. Los requisitos funcionales se satisfacen asignando una secuencia adecuada de responsabilidades a lo largo del diseño. Asignar responsabilidades a los elementos arquitectónicos es fundamental decisión de diseño arquitectónico.
2. Los requisitos de atributos de calidad son satisfechos por las diversas estructuras diseñadas en la arquitectura y los comportamientos e interacciones de los elementos que pueblan esas estructuras.

3. Las restricciones se satisfacen al aceptar la decisión de diseño y conciliarla con otras decisiones de diseño afectadas.

4.2 Funcionalidad

La funcionalidad es la capacidad del sistema para realizar el trabajo para el que fue diseñado. De todos los requisitos, la funcionalidad tiene la relación más extraña con arquitectura. En primer lugar, la funcionalidad no determina la arquitectura. Es decir, dado un conjunto de funcionalidades requeridas, no hay fin para las arquitecturas que puede crear para satisfacer esa funcionalidad. Como mínimo, podría dividir la funcionalidad de varias formas y asigne las subpiezas a diferentes elementos.
De hecho, si la funcionalidad fuera lo único que importara, no habría que dividir el sistema en pedazos; una sola mancha monolítica sin interna la estructura funcionaría bien. En cambio, diseñamos nuestros sistemas como conjuntos estructurados de elementos arquitectónicos cooperantes: módulos, capas, clases, servicios, bases de datos, aplicaciones, hilos, pares, niveles, etc., para hacerlos comprensibles y para apoyar una variedad de otros propósitos. Esos "otros propósitos" son los otros atributos de calidad.
Pero aunque la funcionalidad es independiente de cualquier estructura en particular, la funcionalidad se logra asignando responsabilidades a los elementos arquitectónicos, resultando en una de las estructuras arquitectónicas más básicas.
Aunque las responsabilidades pueden asignarse arbitrariamente a cualquier módulo, la arquitectura de software restringe esta asignación cuando otros atributos de calidad son importantes.
Por ejemplo, los sistemas se dividen con frecuencia para que varias personas puedan construirlos cooperativamente. El interés del arquitecto en la funcionalidad está en cómo interactúa con y restringe otras cualidades.

4.3 Consideraciones sobre atributos de calidad

Así como las funciones de un sistema no funcionan por sí solas sin la debida consideración de otros atributos de calidad, ni los atributos de calidad se destacan por sí mismos; ellos pertenecen a las funciones del sistema. Si un requisito funcional es "Cuando el usuario presiona el botón verde, aparece el cuadro de diálogo Opciones ", una anotación de control de calidad de rendimiento describa la rapidez con la que aparecerá el cuadro de diálogo; una anotación de control de calidad de disponibilidad describa con qué frecuencia fallará esta función y con qué rapidez se reparará; una anotación de usabilidad de control de calidad podría describir lo fácil que es aprender esta función.

Requerimientos funcionales
Después de más de 15 años de escribir y discutir la distinción entre requisitos funcionales y requisitos de calidad, la definición de funcional los requisitos todavía me eluden. Los requisitos de atributos de calidad están bien definido: el ***rendimiento*** *tiene que ver con el comportamiento temporal del sistema,* ***modificabilidad*** *tiene que ver con la capacidad del sistema para soportar cambios en su comportamiento u otras cualidades después de la implementación inicial, la* ***disponibilidad*** *tiene que ver con la capacidad del sistema para sobrevivir a fallas, etc.*
La función, sin embargo, es mucho más resbaladiza. Un estándar internacional (ISO 25010) define la idoneidad funcional como "la capacidad del software

producto para proporcionar funciones que satisfagan las necesidades declaradas e implícitas cuando el software se utiliza en condiciones específicas ". Es decir, la funcionalidad es la capacidad para proporcionar funciones. Una interpretación de esta definición es que la funcionalidad describe lo que hace el sistema y la calidad describe qué tan bien el sistema cumple su función. Es decir, ***las cualidades son atributos del sistema. y la función es el propósito del sistema****.*

Sin embargo, esta distinción se rompe cuando se considera la naturaleza de algunas de las "funciones". Si la función del software es controlar el motor comportamiento, ¿cómo se puede implementar correctamente la función sin considerar comportamiento de tiempo? ¿Es la capacidad de controlar el acceso requiriendo una combinación de nombre de usuario / contraseña no es una función aunque no es la propósito de cualquier sistema?

Me gusta mucho más el uso de la palabra "responsabilidad" para describir los cálculos. que debe realizar un sistema. Preguntas como "¿Cuáles son las limitaciones de tiempo en ese conjunto de responsabilidades? "," ¿Qué modificaciones son anticipado con respecto a ese conjunto de responsabilidades? ", y" ¿Qué clase de los usuarios pueden ejecutar ese conjunto de responsabilidades? " tiene sentido y son accionables.

El logro de cualidades induce responsabilidad; piensa en el usuario ejemplo de nombre / contraseña recién mencionado. Además, se pueden identificar responsabilidades como asociado con un conjunto particular de requisitos.

*Entonces, ¿esto significa que el término "****requisito funcional****" no debería ser ¿usó? La gente entiende el término, pero cuando la precisión es deseado,* ***deberíamos hablar de conjuntos de responsabilidades específicas en su lugar.***

Paul Clements ha despotricado durante mucho tiempo contra el uso descuidado del término "No funcional", y ahora es mi turno de despotricar contra el uso descuidado de la término "funcional", probablemente igualmente ineficaz.

—LB

Desde una perspectiva del arquitecto, **hay tres problemas** con las discusiones previas sobre el sistema de **atributos de calidad**:

1. Las definiciones proporcionadas para un atributo no se pueden probar. No tiene sentido decir que un sistema será "modificable". Cada sistema puede ser modificable con respecto a un conjunto de cambios y no modificable con respecto a otro. Los otros atributos de calidad son similares a esto: un sistema puede ser robusto con respecto a algunas fallas y frágil con respecto a otras. Y así sucesivamente.
2. La discusión a menudo se centra en la cualidad a la que pertenece una preocupación en particular.

¿Es una falla del sistema debido a un ataque de denegación de servicio un aspecto de la disponibilidad, un aspecto del rendimiento, un aspecto de la seguridad o un aspecto de la usabilidad?

Las cuatro comunidades de atributos reclamarían la propiedad de una falla del sistema debido a un ataque de denegación de servicio. Todos son, hasta cierto punto, correctos. Pero esto no nos ayuda, como arquitectos, a comprender y crear soluciones arquitectónicas para gestionar los atributos de interés.
3. Cada comunidad de atributos ha desarrollado su propio vocabulario.
La comunidad del **desempeño** tiene "eventos" que llegan a un sistema, la comunidad de **seguridad** tiene "ataques" llegando a un sistema, la comunidad de **disponibilidad** ha "Fallas" de un sistema, y la comunidad de **usabilidad** tiene "aportes del usuario". Todos de estos pueden referirse a la misma ocurrencia, pero se describen usando términos diferentes.

Una solución a los dos primeros de estos problemas (definiciones no comprobables y preocupaciones superpuestas) es utilizar escenarios de atributos de calidad como un medio para caracterizar atributos de calidad .Una solución al tercer problema es proporcionar una discusión de cada atributo, concentrándose en sus preocupaciones subyacentes, para ilustrar los conceptos que son fundamentales para esa comunidad de atributos.

Hay **dos categorías de atributos de calidad** en los que nos centramos. El primero es aquellos que describen alguna propiedad del sistema en **tiempo de ejecución**, como la **disponibilidad, rendimiento o usabilidad**. El segundo son los que describen alguna **propiedad del desarrollo del sistema**, como m**odificabilidad** o capacidad de prueba.
Dentro de los sistemas complejos, los atributos de calidad nunca se pueden lograr de forma aislada.
El logro de cualquiera tendrá un efecto, a veces positivo y a veces negativo, sobre el logro de otros. Por ejemplo, casi todos los atributos de calidad afectan negativamente al rendimiento. Toma la portabilidad. La técnica principal para lograr software portátil es aislar las dependencias del sistema, que introduce gastos generales en la ejecución del sistema, generalmente como proceso o procedimiento límites, y esto perjudica el rendimiento. Determinando el diseño que satisface todos los requisitos de atributos de calidad es en parte una cuestión de hacer compensaciones apropiadas.

4.4 Especificación de los requisitos de los atributos de calidad

Un requisito de atributo de calidad debe ser inequívoco y comprobable. Usamos un forma común para especificar todos los requisitos de atributos de calidad. Esto tiene la ventaja de enfatizar los puntos en común entre todos los atributos de calidad. Tiene la desventaja de ser ocasionalmente un ajuste forzado para algunos aspectos de los atributos de calidad.
Nuestra forma común de expresión de atributos de calidad tiene estas partes:
■**Estímulo**. Usamos el término "estímulo" para describir un evento que llega al sistema. El estímulo puede ser un evento para la comunidad de espectáculos, un operación del usuario a la comunidad de usabilidad, o un ataque a la seguridad comunidad. Usamos el mismo término para describir una acción motivadora para el desarrollo de cualidades.Por tanto, un estímulo para la modificabilidad es una solicitud de una modificación; un estímulo para la comprobabilidad es la finalización de una fase de desarrollo.
■ **Fuente de estímulo**. Un estímulo debe tener una fuente, debe provenir de alguna parte.

La fuente del estímulo puede afectar la forma en que el sistema lo trata.
Una solicitud de un usuario de confianza no se someterá al mismo escrutinio que una solicitud de un usuario que no es de confianza.
■ **Respuesta**. La forma en que el sistema debe responder al estímulo también debe ser especificado. La respuesta consiste en las responsabilidades que el sistema (para calidades de tiempo de ejecución) o los desarrolladores (para calidades de tiempo de desarrollo) debe actuar en respuesta al estímulo. Por ejemplo, en una actuación escenario, llega un evento (el estímulo) y el sistema debe procesar ese evento y generar una respuesta. En un escenario de modificabilidad, una solicitud para que llegue una modificación (el estímulo) y los desarrolladores deberían implementar la modificación, sin efectos secundarios, y luego probar e implementar el modificación.
■**Medida de respuesta.** Determinar si una respuesta es satisfactoria-si se cumple el requisito-se habilita al proporcionar una respuesta la medida. Para el rendimiento, esto podría ser una medida de latencia o rendimiento; para la modificabilidad, podría ser la mano de obra o el tiempo de reloj de pared requerido para hacer,probar e implementar la modificación.

Estas cuatro características de un escenario son el corazón de nuestras especificaciones de atributo de calidad. Pero hay dos características más que son importantes: el medio ambiente. y artefacto.
■**Medio ambiente**. El entorno de un requisito es el conjunto de circunstancias en el que tiene lugar el escenario. El medio ambiente actúa como calificador del estímulo. Por ejemplo, una solicitud de modificación que llega después el código se ha congelado para que una versión pueda tratarse de manera diferente a una que llega antes de la helada. Un fracaso que es el quinto fracaso consecutivo de un componente puede tratarse de manera diferente a la primera falla de ese componente.
■**Artefacto**. Finalmente, el artefacto es la parte del sistema a la que se aplica el requisito. Con frecuencia, este es el sistema completo, pero ocasionalmente se pueden llamar partes específicas del sistema. Un fracaso en un almacén de datos puede tratarse de manera diferente a una falla en el almacén de metadatos.
Las modificaciones a la interfaz de usuario pueden tener tiempos de respuesta más rápidos que modificaciones al middleware.

Para resumir cómo especificamos los requisitos de atributos de calidad, capturamos formalmente como escenarios de seis partes. Si bien es común omitir uno o más de estas seis partes, particularmente en las primeras etapas del pensamiento sobre los atributos de calidad,saber que todas las partes están allí obliga al arquitecto a considerar si cada parte es relevante.
En resumen, aquí están las seis partes:
1. Fuente de estímulo. Esta es alguna entidad (un ser humano, un sistema informático o cualquier otro actuador) que generó el estímulo.
2. Estímulo. El estímulo es una condición que requiere una respuesta cuando llega en un sistema.
3. Medio ambiente. El estímulo se produce en determinadas condiciones. El sistema puede estar en una condición de sobrecarga o en funcionamiento normal, o alguna otra estado. Para muchos sistemas, el funcionamiento "normal" puede referirse a uno de varios de modos. Para este tipo de sistemas, el entorno debe especificar en qué modo está ejecutando el sistema.

4. Artefacto. Se estimula algún artefacto. Puede ser una colección de sistemas, todo el sistema, o una parte o partes de él.
5. Respuesta. La respuesta es la actividad realizada como resultado de la llegada. del estímulo.
6. Medida de respuesta. Cuando ocurre la respuesta, debe ser medible en algo de moda para que se pueda probar el requisito.
Distinguimos escenarios de atributos de calidad general (que llamamos escenarios "para abreviar): aquellos que son independientes del sistema y pueden, potencialmente, pertenecen a cualquier sistema, desde escenarios concretos de atributos de calidad (escenarios concretos) -aquellos que son específicos del sistema en particular bajo consideración.
Podemos caracterizar los atributos de calidad como una colección de escenarios generales. Por supuesto, para traducir estas caracterizaciones de atributos genéricos en requisitos para un sistema en particular, los escenarios generales deben ser específicos del sistema.
En Figura 4.1. muestra las partes de un escenario de atributos de calidad que acabamos de discutir. Figura 4.2 muestra un ejemplo de un escenario general, en este caso de disponibilidad.

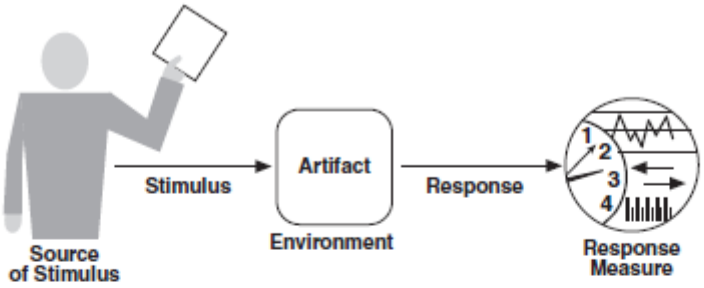

FIGURE 4.1 The parts of a quality attribute scenario

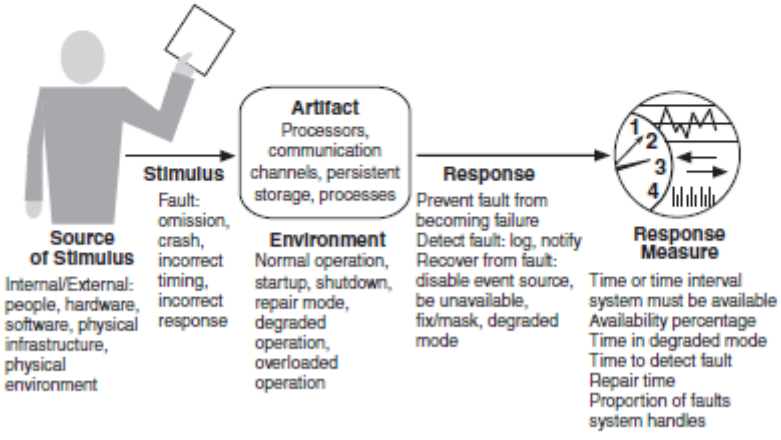

FIGURE 4.2 A general scenario for availability

4.5 Adquisición de atributos de calidad a través de tácticas

A estas técnicas las llamamos tácticas arquitectónicas. Una táctica es una decisión de diseño que influye en el logro de una respuesta de atributo de calidad: tácticas directamente afectan la respuesta del sistema a algún estímulo. Las tácticas imparten portabilidad a uno diseño, alto rendimiento a otro e integrabilidad a un tercero.

El enfoque de una táctica está en una respuesta de atributo de calidad única. Dentro de una táctica no hay consideración de compensaciones. Las compensaciones deben considerarse explícitamente y controlado por el diseñador. A este respecto, las tácticas difieren de los patrones arquitectónicos, donde las compensaciones se integran en el patrón.
El diseño de un sistema consiste en una colección de decisiones. Algunas de estas decisiones ayudan a controlar las respuestas de los atributos de calidad; otros aseguran el logro de funcionalidad del sistema. Representamos la relación entre estímulo, táctica y respuesta en la Figura 4.3. Las tácticas, como los patrones de diseño, son técnicas de diseño que los arquitectos han estado usando durante años. Nuestro aporte es aislar, catalogar, y describirlos. No estamos inventando tácticas aquí, solo estamos capturando lo que los arquitectos hacen en la práctica.
¿Por qué hacemos esto? Hay tres razones:
1. Los patrones de diseño son complejos; normalmente consisten en un paquete de decisiones de diseño. Pero los patrones a menudo son difíciles de aplicar tal cual; los arquitectos necesitan modificarlos y adaptarlos. Al comprender el papel de las tácticas, un arquitecto puede evaluar más fácilmente las opciones para aumentar un patrón existente para lograr un objetivo de atributo de calidad.

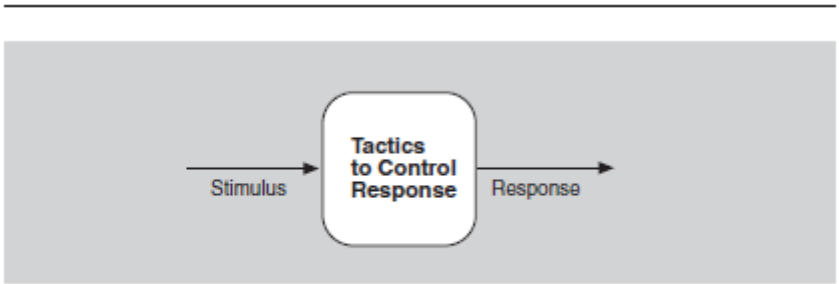

FIGURE 4.3 Tactics are intended to control responses to stimuli.

2. Si no existe un patrón para realizar el objetivo de diseño del arquitecto, las tácticas permiten arquitecto para construir un fragmento de diseño a partir de los "primeros principios". Las tácticas dan conocimiento del arquitecto sobre las propiedades del fragmento de diseño resultante.
3. Al catalogar tácticas, proporcionamos una forma de hacer que el diseño sea más sistemático dentro de algunas limitaciones. Nuestra lista de tácticas no proporciona una taxonomía. Nosotros solo proporcione una categorización. Las tácticas se superpondrán y con frecuencia

tendrá la opción de elegir entre múltiples tácticas para mejorar un atributo de calidad en particular.
La elección de qué táctica a utilizar depende de factores como las compensaciones entre otros atributos de calidad y el costo de implementación. Estas consideraciones trasciende la discusión de tácticas para atributos de calidad particulares.
Las tácticas que presentamos pueden y deben perfeccionarse. Considere el rendimiento: Programar recursos es una táctica de desempeño común. Pero esta táctica necesita para ser refinado en una estrategia de programación específica, como el trabajo más corto primero, la ronda de tareas,y así sucesivamente, para propósitos específicos. Usar un intermediario es una táctica de modificabilidad. Pero hay varios tipos de intermediarios (capas, corredores y proxies,por nombrar unos cuantos). Por lo tanto, hay refinamientos que un diseñador empleará para que cada táctica sea concreta.
Además, la aplicación de una táctica depende del contexto. De nuevo considerando rendimiento: la gestión de la frecuencia de muestreo es relevante en algunos sistemas en tiempo real pero no en todos los sistemas en tiempo real y ciertamente no en los sistemas de bases de datos.