Otros atributos de calidad

La calidad no es un acto, es un hábito.
-Aristóteles

Ya hemos tratado uno a uno con una calidad particular o atributo importante de los sistemas de software. Cada uno de esos temas discutió cómo se define su atributo de calidad particular, dio un escenario general para ese atributo de calidad y mostró cómo escribir escenarios específicos para expresar matices precisos de significado con respecto a ese atributo de calidad. Y cada uno dio una colección de técnicas para lograr ese atributo de calidad en una arquitectura. En resumen, cada tema presentó un tipo de cartera para especificar y diseñar para lograr un atributo de calidad particular.

Esos siete temas cubrieron siete de los atributos de calidad más importantes, en términos de su ocurrencia en los sistemas modernos que dependen del software. Sin embargo, como queda claro, siete solo comienza a rascar la superficie de los atributos de calidad que puede necesitar en un sistema de software en el que esté trabajando.

¿Es el costo un atributo de calidad? No es un atributo de calidad técnica, pero ciertamente afecta la aptitud para el uso. No consideramos los factores económicos.

Este capítulo brindará una breve introducción a algunos otros atributos de calidad, una especie de "lista B" de atributos de calidad, pero, lo que es más importante, muestra cómo construir el mismo tipo de especificación o portafolio de diseño para un atributo de calidad que no está cubierto en nuestro lista.

12.1. OTROS ATRIBUTOS DE CALIDAD IMPORTANTES

Además de los atributos de calidad que hemos tratado en profundidad con los temas anteriores , algunos otros que surgen con frecuencia son la variabilidad, la portabilidad, distributability desarrollo, escalabilidad y elasticidad, capacidad de despliegue, movilidad y posibilidad de seguimiento.

Variabilidad

La variabilidad es una forma especial de modificabilidad. Se refiere a la capacidad de un sistema y sus artefactos de soporte, tales como

requisitos, planes de prueba y especificaciones de configuración para admitir la producción de un conjunto de variantes que difieren entre sí en una forma planificada de antemano. La variabilidad es un atributo de calidad especialmente importante en una línea de productos de software (esto se explorará en profundidad en el Capítulo 25), donde significa la capacidad de un activo central para adaptarse a los usos en los diferentes contextos de productos que se encuentran dentro del alcance de la línea de productos. El objetivo de la variabilidad en una línea de productos de software es facilitar la creación y el mantenimiento de productos en la línea de productos durante un período de tiempo. Los escenarios de variabilidad tratarán el tiempo de unión de la variación y el tiempo de las personas para lograrlo.

Portabilidad

La portabilidad es también una forma especial de modificabilidad. La portabilidad se refiere a la facilidad con la que el software que se creó para ejecutarse en una plataforma se puede cambiar para ejecutarse en una plataforma diferente. La portabilidad se logra al minimizar las dependencias de la plataforma en el software, aislando las dependencias en ubicaciones bien identificadas y escribiendo el software para que se ejecute en una "máquina virtual" (como una Máquina Virtual de Java) que encapsule todas las dependencias de la plataforma. Los escenarios que describen la portabilidad tienen que ver con mover el software a una nueva plataforma al no gastar más de un cierto nivel de esfuerzo o al contar el número de lugares en el software que tendrían que cambiar.

Desarrollo Distributibilidad

La distribución del desarrollo es la calidad del diseño del software para soportar el desarrollo de software distribuido. Muchos sistemas en estos días se desarrollan utilizando equipos distribuidos globalmente. Un problema que debe superarse cuando se desarrolla con equipos distribuidos es coordinar sus actividades. El sistema debe diseñarse de modo que se minimice la coordinación entre los equipos. Esta coordinación mínima debe lograrse tanto para el código como para el modelo de datos. Los equipos que trabajan en módulos que se comunican entre sí pueden necesitar negociar las interfaces de esos módulos. Cuando un módulo es utilizado por muchos otros módulos, cada uno desarrollado por un equipo diferente, la

comunicación y la negociación se vuelven más complejas y onerosas. Se aplican consideraciones similares para el modelo de datos.

Escalabilidad

Dos tipos de escalabilidad son la escalabilidad horizontal y la escalabilidad vertical. La escalabilidad horizontal (ampliación) se refiere a agregar más recursos a las unidades lógicas, como agregar otro servidor a un grupo de servidores. La escalabilidad vertical (ampliación) se refiere a agregar más recursos a una unidad física, como agregar más memoria a una sola computadora. El problema que surge con cualquier tipo de escala es cómo utilizar efectivamente los recursos adicionales. Ser *efectivo*significa que los recursos adicionales resultan en una mejora medible de la calidad del sistema, no requirieron un esfuerzo indebido para agregar y no interrumpieron las operaciones. En entornos de nube, la escalabilidad horizontal se llama *elasticidad*.. La elasticidad es una propiedad que permite a un cliente agregar o eliminar máquinas virtuales del grupo de recursos (consulte el Capítulo 26 para obtener más información sobre dichos entornos). Estas máquinas virtuales están alojadas en una gran colección de más de 10,000 máquinas físicas que son administradas por el proveedor de la nube. Los escenarios de escalabilidad se ocuparán del impacto de agregar o eliminar recursos, y las medidas reflejarán la disponibilidad asociada y la carga asignada a los recursos existentes y nuevos.

Implementación

La implementación se refiere a cómo llega un ejecutable a una plataforma de host y cómo se invoca posteriormente. Algunos de los problemas involucrados en la implementación del software son: ¿Cómo llega a su host (push, donde las actualizaciones se envían a los usuarios de forma no deseada, o pull, donde los usuarios deben solicitar actualizaciones de forma explícita)? ¿Cómo se integra en un sistema existente? ¿Se puede hacer esto mientras se está ejecutando el sistema existente? Los sistemas móviles tienen sus propios problemas en cuanto a cómo se actualizan, debido a las preocupaciones sobre el ancho de banda. Los escenarios de implementación tratarán con el tipo de actualización (push o pull), la forma de la actualización (medio, como la descarga de DVD o Internet, y el empaquetado, como ejecutable, aplicación o complemento), la integración resultante en un

Sistema existente, la eficiencia de ejecución del proceso y el riesgo asociado.

Movilidad

La movilidad se ocupa de los problemas de movimiento y los costos de una plataforma (por ejemplo, tamaño, tipo de pantalla, tipo de dispositivos de entrada, disponibilidad y volumen de ancho de banda y duración de la batería). Los problemas de movilidad incluyen la administración de la batería, la reconexión después de un período de desconexión y la cantidad de diferentes interfaces de usuario necesarias para soportar múltiples plataformas. Los escenarios tratarán con la especificación de los efectos deseados de la movilidad o las diferentes posibilidades. Los escenarios también pueden lidiar con la variabilidad, donde el mismo software se implementa en múltiples plataformas (quizás radicalmente diferentes).

Monitorabilidad

La capacidad de supervisión se relaciona con la capacidad del personal de operaciones para supervisar el sistema mientras se está ejecutando. Los elementos como la longitud de la cola, el tiempo promedio de procesamiento de la transacción y el estado de varios componentes deben ser visibles para el personal de operaciones para que puedan tomar medidas correctivas en caso de problemas potenciales. Los escenarios tratarán un problema potencial y su visibilidad para el operador, y una posible acción correctiva.

La seguridad

En 2009, un empleado de la central hidroeléctrica Shushenskaya en Siberia envió comandos a través de una red para activar, de forma remota, una turbina no utilizada. La turbina fuera de línea creó un "martillo de agua" que inundó y luego destruyó la planta y mató a docenas de trabajadores.

La idea de que el software podría matar a la gente solía pertenecer al ámbito de la ciencia ficción de las computadoras kitsch-run-amok. Lamentablemente, no se quedó allí. A medida que el software ha llegado a controlar cada vez más dispositivos en nuestras vidas, la seguridad del software se ha convertido en una preocupación fundamental.

La seguridad no es solo una cuestión de software, sino una preocupación por cualquier sistema que pueda afectar su

entorno. Como tal, recibe una mención en la Sección 12.3 , donde analizamos los atributos de calidad del sistema. Pero existen medios para abordar la seguridad que están totalmente en el ámbito del software, por lo que también lo discutimos aquí.

La seguridad del software se trata de la capacidad del software para evitar entrar en estados que causan o llevan a daños, lesiones o pérdida de vidas a los actores en el entorno del software, y para recuperar y limitar el daño cuando entra en estados malos. Otra forma de decirlo es que la seguridad se refiere a la prevención y recuperación de fallas peligrosas. Debido a esto, las preocupaciones arquitectónicas con respecto a la seguridad son casi idénticas a las de disponibilidad, que también se trata de evitar y recuperarse de las fallas. Las tácticas de seguridad, entonces, se superponen con las de disponibilidad en gran medida. Ambos incluyen tácticas para prevenir fallas y detectar y recuperarse de fallas que ocurren.

La seguridad no es lo mismo que la fiabilidad. Un sistema puede ser confiable (consistente con su especificación) pero aún inseguro (por ejemplo, cuando la especificación ignora las condiciones que conducen a una acción insegura). De hecho, prestar atención a la especificación del software crítico para la seguridad es quizás lo más poderoso que puede hacer para producir software seguro. Las fallas y los peligros no se pueden detectar, prevenir o mejorar si el software no se ha diseñado pensando en ellos. La seguridad se diseña con frecuencia mediante la realización de análisis de efectos y modos de falla, análisis de peligros y análisis de árbol de fallas. (Estas técnicas se analizan en el Capítulo 5). Estas técnicas tienen como objetivo descubrir posibles peligros que podrían resultar de la operación del sistema y proporcionar planes para hacer frente a estos peligros.

12.2. OTRAS CATEGORÍAS DE ATRIBUTOS DE CALIDAD

En nuestras discusiones sobre atributos de calidad, nos hemos centrado principalmente en las cualidades de los productos, pero existen otros tipos de atributos de calidad que miden la "bondad" de algo distinto del producto final. Aquí hay tres:

Integridad conceptual de la arquitectura

La integridad conceptual se refiere a la consistencia en el diseño de la arquitectura, y contribuye a la comprensión de la arquitectura y conduce a menos errores de confusión. La integridad conceptual exige

que lo mismo se haga de la misma manera a través de la arquitectura. En una arquitectura con integridad conceptual, menos es más. Por ejemplo, hay innumerables formas en que los componentes pueden enviarse información entre sí: mensajes, estructuras de datos, señalización de eventos, etc. Una arquitectura con integridad conceptual se presentaría de una sola manera, y solo proporcionaría alternativas si hubiera una razón convincente para hacerlo. De manera similar, todos los componentes deben informar y manejar los errores de la misma manera, registrar eventos o transacciones de la misma manera, interactuar con el usuario de la misma manera, y así sucesivamente.

Calidad en uso

ISO / IEC 25010, que analizamos más adelante, tiene una categoría de cualidades que pertenecen al uso del sistema por parte de diversos interesados. Por ejemplo, el tiempo de comercialización es una característica importante de un sistema, pero no se puede discernir al examinar el producto en sí. Algunas de las cualidades en esta categoría son estas:

• *Efectividad* . Esto se refiere a la distinción entre construir el sistema correctamente (el sistema funciona de acuerdo con sus requisitos) y construir el sistema correcto (el sistema funciona de la manera que el usuario desea). La efectividad es una medida de si el sistema es correcto.

• *Eficiencia* . El esfuerzo y tiempo requerido para desarrollar un sistema. Dicho de otra manera, ¿cuál es el impacto de la arquitectura en el costo y el cronograma del proyecto? ¿Un conjunto diferente de opciones arquitectónicas hubiera dado lugar a un sistema que sería más rápido o más barato llevar a buen término? La eficiencia puede incluir tiempo de capacitación para los desarrolladores; una arquitectura que utiliza tecnología desconocida para el personal disponible es menos edificable. ¿Es la arquitectura adecuada para la organización en términos de su experiencia y su infraestructura de soporte disponible (como instalaciones de prueba o entornos de desarrollo)?

• *Libertad de riesgo* . El grado en que un producto o sistema afecta el estado económico, la vida humana, la salud o el medio ambiente.

Un caso especial de eficiencia es lo fácil que es construir (es decir, compilar y ensamblar) el sistema después de un cambio. Esto se vuelve

crítico durante la prueba. Un proceso de recompilación que lleva horas o durante la noche es un asesino de programación. Los arquitectos tienen control sobre esto al administrar las dependencias entre los módulos. Si el arquitecto no hace esto, entonces lo que sucede a menudo es que un desarrollador de ojos brillantes escribe un archivo make desde el principio, funciona, y la gente lo agrega y lo agrega. Finalmente, el proyecto termina con un paso de compilación de siete horas e integradores y evaluadores muy descontentos que ya están atrasados (porque siempre lo están).

Comerciabilidad

La comercialización de una arquitectura es otro atributo de calidad que preocupa. Algunos sistemas son bien conocidos por sus arquitecturas, y estas arquitecturas a veces tienen un significado propio, independientemente de los otros atributos de calidad que aportan al sistema. La moda actual en la construcción de sistemas basados en la nube nos ha enseñado que la percepción de una arquitectura puede ser más importante que las cualidades que aporta la arquitectura. Muchas organizaciones han sentido que tenían que construir sistemas basados en la nube (o alguna otra *tecnología du jour*) si esa era la elección técnica correcta o no.

12.3. ATRIBUTOS DE CALIDAD DEL SOFTWARE Y ATRIBUTOS DE CALIDAD DEL SISTEMA

Los sistemas físicos, como aeronaves o automóviles o aparatos de cocina, que se basan en el software incorporado, están diseñados para cumplir con toda una letanía de atributos de calidad: peso, tamaño, consumo eléctrico, potencia de salida, salida de contaminación, resistencia a la intemperie, vida útil de la batería y incesantemente. Para muchos de estos sistemas, la seguridad encabeza la lista (consulte la barra lateral).

A veces, la arquitectura del software puede tener un efecto sorprendente en los atributos de calidad del sistema. Por ejemplo, el software que hace un uso ineficiente de los recursos informáticos puede requerir memoria adicional, un procesador más rápido, una batería más grande o incluso un procesador adicional. Los procesadores adicionales pueden aumentar el consumo de energía, el peso, el espacio requerido en el gabinete y, por supuesto, los gastos del sistema.

La computación ecológica es un tema de creciente preocupación. Recientemente hubo una controversia sobre la cantidad de gases de efecto invernadero que se bombeaban a la atmósfera por las granjas de procesadores masivos de Google. Dada la producción diaria y la cantidad de solicitudes diarias, es posible estimar la cantidad de gases de efecto invernadero que *usted* emite cada vez que *le* pide a Google que realice una búsqueda. (Las estimaciones actuales oscilan entre 0,2 gramos y 7 gramos de CO_2). La computación verde es la furia. Eve Troeh, en el programa "Marketplace" (5 de julio de 2011) de American Public Media, informa:

Según la Agencia de Protección Ambiental, el dos por ciento de toda la electricidad de los Estados Unidos ahora va a los centros de datos. La electricidad se ha convertido en el mayor costo para procesar datos, más que el equipo para hacerlo, más que los edificios para albergar ese equipo. . . . Google está haciendo servidores de datos que pueden flotar en alta mar, enfriados por brisas oceánicas. HP tiene planes de colocar servidores de datos cerca de las granjas y alimentarlos con gas metano proveniente de pasteles de vaca.

La lección aquí es que si usted es el arquitecto de software que reside en un sistema más grande, deberá comprender los atributos de calidad que son importantes para el sistema de contención y trabajar con los arquitectos e ingenieros del sistema para ver cómo funciona su software. La arquitectura puede contribuir a conseguirlos.

La línea de fuga entre el software y las cualidades del sistema

Este es un libro sobre arquitectura de software, por lo que tratamos los atributos de calidad desde la perspectiva de un arquitecto de software. Pero es posible que ya haya notado que los atributos de calidad que el arquitecto de software puede aportar a la parte están limitados por la arquitectura del sistema en el que se ejecuta el software.

Por ejemplo:

• El rendimiento de una pieza de software está fundamentalmente limitado por el rendimiento de la computadora que lo ejecuta. No importa qué tan bien diseñe el software, simplemente no puede ejecutar los últimos modelos de predicción meteorológica de toda la Tierra en el Commodore 64 de Grampa y espero saber si va a llover mañana.

• La seguridad física es probablemente más importante y más efectiva que la seguridad del software para prevenir el fraude y el robo. Si no cree esto, escriba la contraseña de su computadora portátil en un pedazo de papel, péguela en su computadora portátil y déjela en un auto sin llave con las ventanas abiertas. (En realidad, no hagas eso. Considera esto como un experimento mental).

• Si estamos siendo perfectamente honestos aquí, ¿qué tan útil es un dispositivo para la navegación web que tiene una pantalla más pequeña que una tarjeta de crédito y que las llaves son del tamaño de una pasa?

Para mí, en ninguna parte la barrera entre el software y el sistema es más nebulosa que en el área de seguridad. La idea de que los programas (cadenas de 0 y 1) pueden matar, mutilar o destruir sigue siendo una noción antinatural. Por supuesto, no son los 0 y los 1 los que causan estragos. Al menos, no directamente. Es a lo que están conectados. El software, y el sistema en el que se ejecuta, deben conectarse al mundo exterior de alguna manera antes de que pueda causar daños. Esa es la buena noticia. La mala noticia es que la buena noticia no es tan buena. El software *está* conectado al mundo exterior, siempre. Si tu programano tiene efecto alguno que sea observable fuera de sí mismo, probablemente no sirve para nada.

Hay ejemplos notorios de fallas relacionadas con el software. La catástrofe de la planta hidroeléctrica de Siberia mencionada en el texto, la sobredosis mortal de radiación de Therac-25, la explosión del Ariane 5 y un centenar de accidentes menos conocidos causaron daños porque el *software* era parte de un *sistema*.que incluía una turbina, un emisor de rayos X o los controles de dirección de un cohete, en los ejemplos que acabamos de citar. En estos casos, el software defectuoso ordenó a algunos hardware del sistema que realizaran una acción desastrosa, y el hardware simplemente obedeció. Los actuadores son dispositivos que conectan hardware a software; son el puente entre el mundo de 0 y 1 y el mundo de movimiento y control. Envíe un valor digital a un actuador (o escriba una cadena de bits en el registro de hardware correspondiente al actuador) y ese valor se traduce a alguna acción mecánica, para bien o para mal.

Pero la conexión a un actuador no es necesaria para desastres relacionados con el software. A veces, todo lo que la computadora tiene que hacer es enviar información errónea a sus operadores humanos. En septiembre de 1983, un satélite soviético envió datos a su computadora del sistema terrestre, que interpretó esos datos como un misil lanzado desde los Estados Unidos hacia Moscú. Segundos después, la computadora reportó un segundo misil en vuelo. Pronto, un tercero, luego un cuarto, y luego un quinto apareció. El teniente coronel de las Fuerzas Estratégicas Soviéticas Stanislav Yevgrafovich Petrov tomó la sorprendente decisión de ignorar el sistema de alerta, creyendo que estaba equivocado. Pensó que era extremadamente improbable que los EE. UU. Hubieran disparado solo unos pocos misiles, invitando así a la destrucción total en represalia. Decidió esperar, para ver si los misiles eran reales, es decir, Para ver si la capital de su país iba a ser incinerada. Como sabemos, no lo fue. El sistema soviético había confundido una rara condición de luz solar con misiles en vuelo. Errores similares han ocurrido en el lado estadounidense.

Por supuesto, los humanos no siempre lo hacen bien. En la noche oscura y tormentosa del 1 de junio de 2009, el vuelo 447 de Air France desde Río de Janeiro a París se desplomó en el Océano Atlántico, matando a todos a bordo. Los registradores de vuelo del Airbus A-330 no se recuperaron hasta

mayo de 2011, y cuando este libro se publica, parece que los pilotos nunca supieron que el avión había entrado en un puesto de gran altitud. Los sensores que miden la velocidad del aire se habían obstruido con hielo y, por lo tanto, no eran confiables. Se requirió que el software desconectara el piloto automático en esta situación, y así fue. Los pilotos humanos pensaron que el avión iba demasiado rápido (y en peligro de falla estructural) cuando, de hecho, iba demasiado lento (y se estaba cayendo). Durante la inmersión total de más de tres minutos desde 38,000 pies, los pilotos intentaron tirar de la nariz hacia arriba y aceleraron para bajar la velocidad. Es una buena apuesta que aumentar la confusión era la forma en que funcionaba el sistema de advertencia de bloqueo del A-330. Cuando el sistema detecta un bloqueo, emite una alarma sonora fuerte. Las computadoras desactivan la advertencia de bloqueo cuando "piensan" que las medidas del ángulo de ataque no son válidas. Esto puede ocurrir cuando las lecturas de velocidad del aire son muy bajas. Eso es exactamente lo que sucedió con Air France 447: su velocidad de avance descendió a menos de 60 nudos y el ángulo de ataque fue extremadamente alto. Como consecuencia de una regla en el control de vuelo. Las computadoras desactivan la advertencia de bloqueo cuando "piensan" que las medidas del ángulo de ataque no son válidas. Esto puede ocurrir cuando las lecturas de velocidad del aire son muy bajas. Eso es exactamente lo que sucedió con Air France 447: su velocidad de avance descendió a menos de 60 nudos y el ángulo de ataque fue extremadamente alto. Como consecuencia de una regla en el control de vuelo. Las computadoras desactivan la advertencia de bloqueo cuando "piensan" que las medidas del ángulo de ataque no son válidas. Esto puede ocurrir cuando las lecturas de velocidad del aire son muy bajas. Eso es exactamente lo que sucedió con Air France 447: su velocidad de avance descendió a menos de 60 nudos y el ángulo de ataque fue extremadamente alto. Como consecuencia de una regla en el control de vuelo.*software* , la advertencia de bloqueo se detuvo y comenzóvarias veces. Peor aún, se encendía cada vez que el piloto dejaba caer la nariz (aumentando la velocidad del aire y tomando las lecturas en el rango "válido", pero aún en reposo) y luego se detenía cuando se retiraba. Es decir, hacer lo correcto resultó en una retroalimentación incorrecta y viceversa.

¿Fue este un sistema inseguro, o un sistema seguro operado de manera insegura? En última instancia los tribunales decidirán.

El software que puede dañarnos físicamente es un hecho de nuestra vida moderna. A veces, el vínculo entre el software y el daño físico es directo, como en el ejemplo de Ariane, y otras veces es mucho más tenue, como en el ejemplo de Air France 447. Pero como profesionales de software, no podemos refugiarnos en el hecho de que nuestro software no puede infligir daño más que la persona que grita "¡Fuego!" En un teatro lleno de gente puede afirmar que fue la estampida, no el grito, lo que causó lesiones .

- *PCC*

12.4. USAR LISTAS ESTÁNDAR DE ATRIBUTOS DE CALIDAD, O NO

Los arquitectos no tienen escasez de listas de atributos de calidad para los sistemas de software a su disposición. El estándar con el título de pausa y toma el aliento de “ISO / IEC FCD 25010: Ingeniería de sistemas y software: requisitos y evaluación de calidad de los sistemas y productos de software (SQuaRE): modelos de sistemas y de calidad de software” es un buen ejemplo. . El estándar divide los atributos de calidad en aquellos que soportan un modelo de "calidad en uso" y aquellos que soportan un modelo de "calidad de producto". Esa división es un poco exagerada en algunos lugares, pero sin embargo comienza una marcha de dividir y conquistar a través de una impresionante variedad de cualidades. Vea la Figura 12.1 para esta matriz.

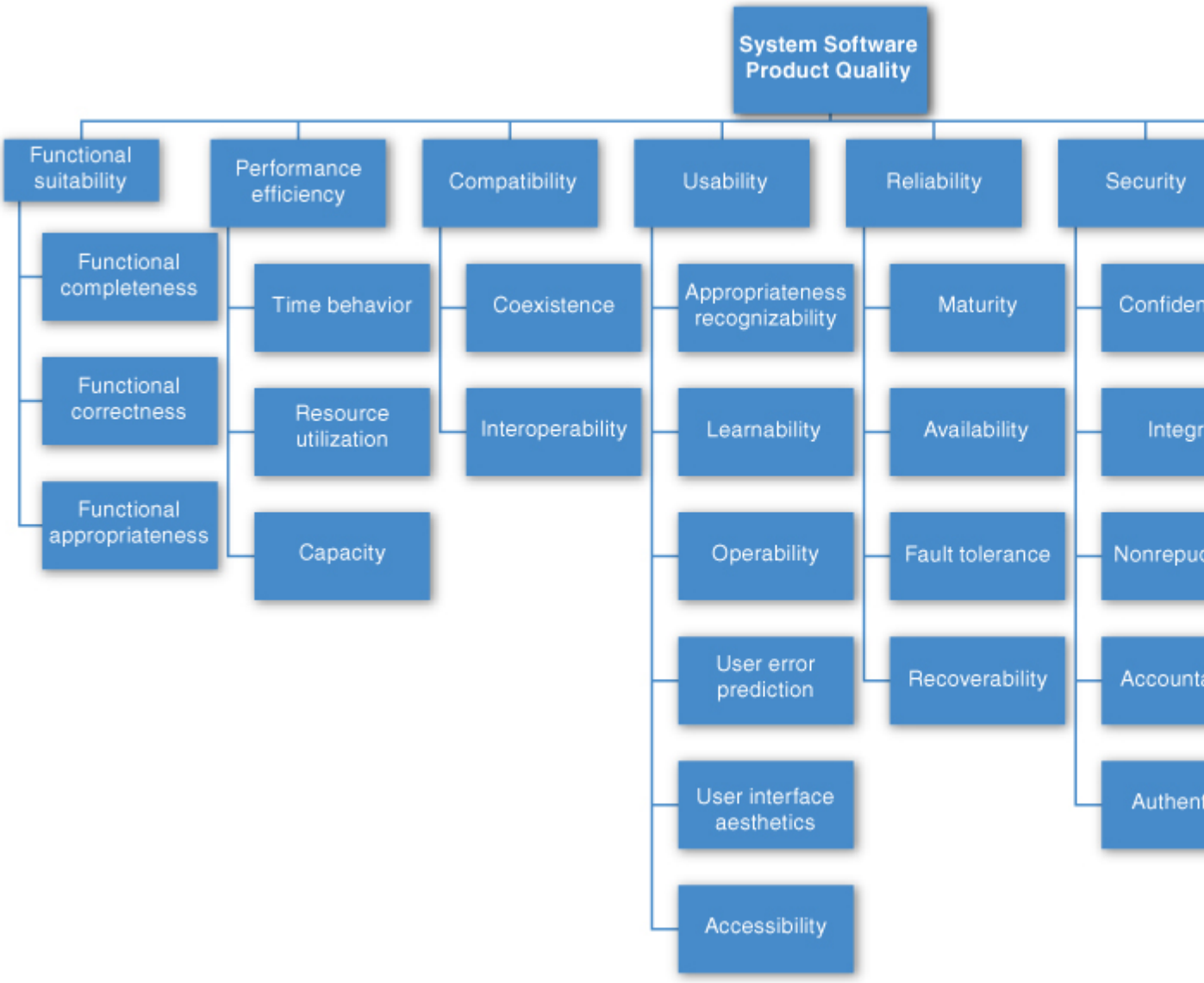

Figura 12.1. El estándar de calidad de producto ISO / IEC FCD 25010.

El estándar enumera los siguientes atributos de calidad que tratan con la calidad del producto:

• *Adecuación funcional* . El grado en que un producto o sistema proporciona funciones que satisfacen necesidades declaradas e implícitas cuando se utilizan en condiciones específicas

• *Eficiencia de rendimiento* . Desempeño relativo a la cantidad de recursos utilizados en las condiciones establecidas.

• *Compatibilidad* . El grado en que un producto, sistema o componente puede intercambiar información con otros productos, sistemas o

componentes, y / o desempeñar sus funciones requeridas, mientras comparte el mismo entorno de hardware o software

• *Usabilidad* . El grado en que un producto o sistema puede ser utilizado por usuarios específicos para lograr objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso específico

• *Fiabilidad* . El grado en que un sistema, producto o componente realiza funciones específicas bajo condiciones específicas durante un período de tiempo específico

• *Seguridad* . El grado en que un producto o sistema protege la información y los datos para que las personas u otros productos o sistemas tengan el grado de acceso a los datos adecuado a sus tipos y niveles de autorización.

• *Mantenibilidad* . El grado de efectividad y eficiencia con el cual un producto o sistema puede ser modificado por los mantenedores previstos

• *Portabilidad* . El grado de efectividad y eficiencia con el que un sistema, producto o componente puede transferirse de un hardware, software u otro entorno operativo o de uso a otro

En la norma ISO 25010, estas "características de calidad" están compuestas de "características secundarias de calidad" (por ejemplo, el no repudio es una característica secundaria de la seguridad). Los slogs estándar a través de casi cinco docenas de descripciones separadas de las subcaracterísticas de calidad de esta manera. Define para nosotros las cualidades de "placer" y "comodidad". Distingue entre "corrección funcional" y "integridad funcional", y luego agrega "adecuación funcional" para una buena medida. Para exhibir "compatibilidad", los sistemas deben tener "interoperabilidad" o simplemente "coexistencia". "Usabilidad" es una calidad del producto, no una calidad de uso, aunque incluye "satisfacción", que *es*Una calidad de uso en calidad. La "modificabilidad" y la "comprobabilidad" son parte de la "mantenibilidad". También lo es la "modularidad", que es una estrategia para lograr una calidad en lugar de una meta en sí misma. "Disponibilidad" es parte de "confiabilidad". "Interoperabilidad" es parte de "compatibilidad". Y "escalabilidad" no se menciona en absoluto.

¿Tienes todo eso?

Listas como éstas, y hay muchas, sirven un propósito. Pueden ser listas de verificación útiles para ayudar a los recolectores de requisitos a

asegurarse de que no se pasen por alto las necesidades importantes. Incluso más útiles que las listas independientes, pueden servir como base para crear su propia lista de verificación que contiene los atributos de calidad de interés en su dominio, su industria, su organización y sus productos. Las listas de atributos de calidad también pueden servir como base para establecer medidas. Si el “placer” resulta ser una preocupación importante en su sistema, ¿cómo lo mide para saber si su sistema está proporcionando suficiente cantidad?

Sin embargo, las listas generales como éstas también tienen inconvenientes. En primer lugar, ninguna lista estará completa. Como arquitecto, se le pedirá que diseñe un sistema para satisfacer una inquietud de las partes interesadas no prevista por ningún creador de listas. Por ejemplo, algunos escritores hablan de "capacidad de administración", que expresa lo fácil que es para los administradores de sistemas administrar la aplicación. Esto se puede lograr insertando instrumentos útiles para la operación de monitoreo y para la depuración y el ajuste del rendimiento. Sabemos de una arquitectura que fue diseñada con el objetivo consciente de retener al personal clave y atraer nuevas contrataciones con talento a una región tranquila del Medio Oeste estadounidense. Los arquitectos de ese sistema hablaron de impregnar al sistema con la "capacidad de pago". Lo lograron al incorporar tecnología de punta y dar a sus equipos de desarrollo una amplia libertad creativa.lista estándar de atributos de calidad, pero ese control de calidad era tan importante para esa organización como cualquier otro.

En segundo lugar, las listas a menudo generan más controversia que comprensión. Podría argumentar persuasivamente que la "corrección funcional" debe ser parte de la "confiabilidad", o que la "portabilidad" es solo un tipo de "modificabilidad", o que la "mantenibilidad" es un tipo de "modificabilidad" (no al revés) . Los escritores de ISO 25010 aparentemente pasaron tiempo y esfuerzo decidiendo hacer de la seguridad su propia característica, en lugar de una característica subcaracterística, que era en una versión anterior. Creemos que el esfuerzo para hacer estos argumentos podría ser mejor gastado en otros lugares.

En tercer lugar, estas listas a menudo pretenden ser *taxonomías* , que son listas con la propiedad especial que cada miembro puede asignar a un solo lugar. Los atributos de calidad son notoriamente blandos a este respecto. Discutimos la denegación de servicio como parte de la

seguridad, la disponibilidad, el rendimiento y la facilidad de uso en el Capítulo 4 .

Finalmente, estas listas obligan a los arquitectos a prestar atención a cada atributo de calidad en la lista, incluso aunque solo sea para decidir finalmente que el atributo de calidad en particular es irrelevante para su sistema. Saber cómo decidir rápidamente que un atributo de calidad es irrelevante para un sistema específico es una habilidad que se gana con el tiempo.

Estas observaciones refuerzan la lección introducida en el Capítulo 4 de que los nombres de atributos de calidad, en sí mismos, son en gran medida inútiles y, en el mejor de los casos, invitaciones para comenzar una conversación; que dedicar tiempo a preocuparse por qué cualidades son sub-calidades de lo que otras cualidades también es casi inútil; y esos escenarios nos proporcionan la mejor manera de especificar con precisión lo que queremos decir cuando hablamos de un atributo de calidad.

Use listas estándar de atributos de calidad en la medida en que sean útiles como listas de verificación, pero no sienta la necesidad de adherirse servilmente a su terminología.

12.5. LIDIANDO CON LA "HABILIDAD X": TRAER UN NUEVO ATRIBUTO DE CALIDAD AL PLIEGUE

Supongamos, como arquitecto, debe lidiar con un atributo de calidad para los que no hay un cuerpo compacto del conocimiento “cartera”, no como los capítulos 5 - 11 previstas esas siete evaluaciones de calidad? ¿Supongamos que tiene que lidiar con un atributo de calidad como "informática ecológica" o "capacidad de administración" o incluso "capacidad de pago"? ¿Qué haces?

Capturar escenarios para el nuevo atributo de calidad

Lo primero que debe hacer es entrevistar a las partes interesadas cuyas inquietudes han llevado a la necesidad de este atributo de calidad. Puede trabajar con ellos, individualmente o en grupo, para construir un conjunto de caracterizaciones de atributos que refinen lo que se entiende porEl QA. Por ejemplo, la seguridad a menudo se descompone en preocupaciones como la confidencialidad, integridad, disponibilidad y otras. Después de ese refinamiento, puede trabajar con las partes interesadas para elaborar un conjunto de escenarios específicos que caracterizan lo que significa ese control de calidad.

Una vez que tenga un conjunto de escenarios específicos, puede trabajar para generalizar la colección. Observe el conjunto de estímulos que ha recopilado, el conjunto de respuestas, el conjunto de medidas de respuesta, etc. Utilícelas para construir un escenario general haciendo de cada parte del escenario general una generalización de las instancias específicas que recopiló.

En nuestra experiencia, los pasos descritos hasta ahora tienden a consumir aproximadamente medio día.

Ensamblar enfoques de diseño para el nuevo atributo de calidad

Después de tener un conjunto de escenarios de guía para el control de calidad, puede ensamblar un conjunto de enfoques de diseño para enfrentarlo. Puedes hacer esto por

1. Revisar un conjunto de patrones con los que está familiarizado y preguntarse cómo afecta cada uno al QA de interés.

2. Buscar diseños que hayan tenido que lidiar con este control de calidad. Puede buscar en el nombre que le dio al QA, pero también puede buscar los términos que eligió cuando refinó el QA en caracterizaciones de atributos subsidiarios (como "confidencialidad" para el QA de seguridad).

3. Encontrar expertos en esta área y entrevistarlos o simplemente escribir y pedirles consejo.

4. Usar el escenario general para tratar de catalogar una lista de enfoques de diseño para producir las respuestas en la categoría de respuesta.

5. Usar el escenario general para catalogar una lista de formas en que una arquitectura problemática no podría producir las respuestas deseadas, y pensar en enfoques de diseño para evitar esos casos.

Modelar el nuevo atributo de calidad

Si puede construir un modelo conceptual del atributo de calidad, esto puede ser útil para crear un conjunto de enfoques de diseño para él. Por "modelo", no queremos decir nada más que entender el conjunto de parámetros a los que el atributo de calidad es sensible. Por ejemplo, un modelo de modificabilidad podría decirnos que la modificabilidad es una función de cuántos lugares en un sistema se deben cambiar en respuesta a una modificación y la interconexión de esos lugares. Un modelo de rendimiento podría decirnos que el rendimiento es una función de la carga de trabajo transaccional, las

dependencias entre las transacciones y la cantidad de transacciones que se pueden procesar en paralelo.

Una vez que tenga un modelo para su control de calidad, puede trabajar para catalogar los enfoques arquitectónicos (tácticas y patrones) abiertos para manipular cada uno de los parámetros relevantes a su favor.

Arme un conjunto de tácticas para el nuevo atributo de calidad

Hay dos fuentes que pueden usarse para derivar tácticas para cualquier atributo de calidad: modelos y expertos.

La figura 12.2 muestra un modelo de cola para el rendimiento. Dichos modelos se utilizan ampliamente para analizar la latencia y el rendimiento de varios tipos de sistemas de colas, incluidos los entornos de fabricación y servicio, así como los sistemas informáticos.

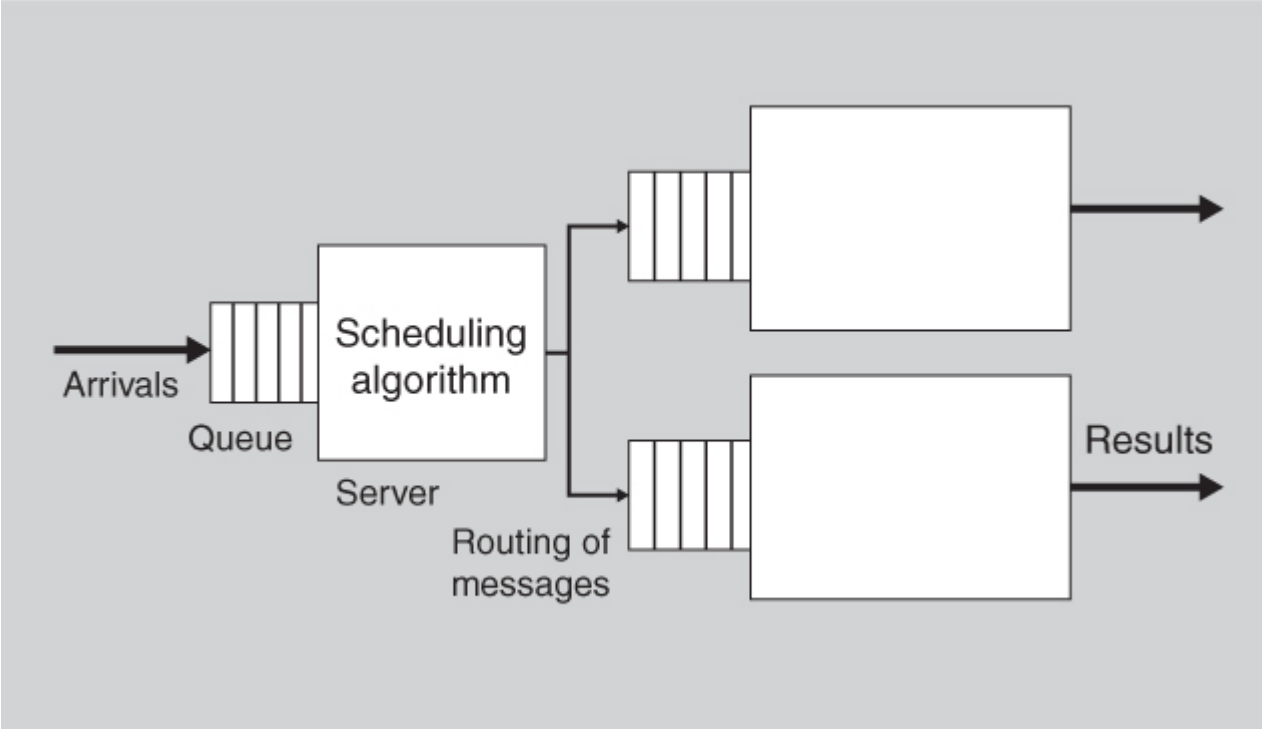

Figura 12.2. Un modelo genérico de colas.

Dentro de este modelo, hay siete parámetros que pueden afectar la latencia que el modelo predice:

- Tasa de llegada
- disciplina de colas
- algoritmo de programación

• Tiempo de servicio
• Topología
• Ancho de banda de la red
• Algoritmo de enrutamiento

Estos son los *únicos* parámetros que pueden afectar la latencia dentro de este modelo. Esto es lo que le da al modelo su poder. Además, cada uno de estos parámetros puede verse afectado por varias decisiones arquitectónicas. Esto es lo que hace que el modelo sea útil para un arquitecto. Por ejemplo, el algoritmo de enrutamiento puede ser fijo o podría ser un algoritmo de equilibrio de carga. Se debe elegir un algoritmo de programación. La topología puede verse afectada por la adición o eliminación dinámica de nuevos servidores. Etcétera.

El proceso de generación de tácticas basadas en un modelo es este:

• Enumerar los parámetros del modelo.
• Para cada parámetro, enumere las decisiones arquitectónicas que pueden afectar este parámetro

Lo que resulta es una lista de tácticas para, en el caso de ejemplo, controlar el rendimiento y, en el caso más general, para controlar el atributo de calidad que concierne al modelo. Esto hace que el problema de diseño parezca mucho más manejable. Esta lista de tácticas es finita y razonablemente pequeña, porque la cantidad de parámetros del modelo está limitada, y para cada parámetro, la cantidad de decisiones arquitectónicas que afectan el parámetro es limitada.

Las tácticas de derivación de modelos están bien siempre que el atributo de calidad en cuestión tenga un modelo. Desafortunadamente, el número de tales modelos es limitado y es un tema de investigación activa. No hay buenos modelos *arquitectónicos* de usabilidad o seguridad, por ejemplo. En los casos en los que no teníamos un modelo para trabajar, hicimos cuatro cosas para catalogar las tácticas:

1. Entrevistamos a expertos en el campo, preguntándoles qué hacen como arquitectos para mejorar la respuesta del atributo de calidad.

2. Examinamos los sistemas que se promocionaron como de alta usabilidad (o probabilidad, o cualquier táctica en la que nos estuviéramos enfocando).

3. Recorrimos la literatura de diseño relevante en busca de temas comunes en el diseño.

4. Examinamos los patrones arquitectónicos documentados para buscar formas en que lograron las respuestas de atributos de calidad que se les presentaron.

Construir listas de verificación de diseño para el nuevo atributo de calidad

Finalmente, examine las siete categorías de decisiones de diseño en el Capítulo 4 y pregúntese a sí mismo (oa sus expertos) cómo especializar su nueva calidad de interés para estas categorías. En particular, piense en revisar una arquitectura de software y tratar de averiguar qué tan bien satisface sus nuevas cualidades en estas siete categorías. ¿Qué preguntas le haría al arquitecto de ese sistema para comprender cómo el diseño intenta lograr la nueva calidad? Estas son las bases para la lista de verificación de diseño.