

SQA: Aseguramiento de la Calidad del Software

Ingeniería de Software – IIC2143

Resumen

Conjunto de actividades para asegurar que el software cumplirá requisitos funcionales y de calidad:

- es importante poner atención a la calidad del proceso de desarrollo de software, a cómo se lleva acabo el proceso
- ... y a la calidad de los artefactos producidos durante el desarrollo: que sean correctos, completos, inteligibles

Estas actividades son actividades del ciclo de vida, llevadas a cabo paralelamente a las actividades de desarrollo

Beneficios

Los errores en el software pueden costar billones de dólares debido a daños a la propiedad, pérdida de productividad, daños a las personas, y hasta pérdida de vidas

Atributos de calidad

La calidad del software es determinada por un número de factores, o atributos, de calidad:

- un atributo describe o caracteriza un aspecto del software

Un atributo de calidad describe o caracteriza un aspecto relativo a la calidad del software —típicamente, requisitos no funcionales:

- confiabilidad, robustez, eficiencia, interoperabilidad, mantenibilidad, *testability*, portabilidad, reusabilidad, modularidad, cohesión, acoplamiento
- ISO e IEEE han desarrollado sus respectivos modelos de calidad

Una especificación cualitativa de un requisito no funcional —p.ej., “el sistema debe ser confiable”— no es suficiente:

- necesitamos mediciones y métricas de calidad

Mediciones y métricas

Una medición de software es una evaluación objetiva y cuantitativa de un atributo del software:

- p.ej., el tamaño de un módulo

Una métrica de software es una medición estándar (aunque no hay mediciones estándares oficiales):

- hay métricas para procesos, para proyectos, y para productos

Un indicador es un valor de medición o de métrica, que se considera que tiene una implicación importante :

- p.ej., una función es considerada demasiado compleja si contiene diez o más condiciones binarias atómicas
- son útiles como señales de advertencia y como objetivos de calidad

Utilidad de las mediciones y métricas de calidad

Definirlas y recolectar los datos para calcularlas consume tiempo y recursos y demanda un esfuerzo considerable ... sin embargo

Son evaluaciones cuantitativas del software, incluyendo especificación de requisitos, diseño, implementación, y documentación:

- definición y uso de indicadores
- asignación de recursos valiosos a áreas críticas
- comparación cuantitativa de proyectos y sistemas similares
- evaluación cuantitativa de mejoras
- evaluación cuantitativa de la tecnología
- evaluación cuantitativa de mejoramiento de procesos

Métricas de calidad convencionales y oo

De requisitos:

- no ambigüedad, completitud, corrección, consistencia

De diseño:

- fan-in, fan-out, cohesión, acoplamiento, modularidad, complejidad

De implementación y del sistema:

- líneas de código, complejidad ciclomática, confiabilidad, disponibilidad

Orientadas a objetos:

- complejidad ponderada de los métodos de una clase, profundidad del árbol de herencia, acoplamiento entre clases

Técnicas de verificación y validación

La verificación asegura que el proceso es ejecutado correctamente

La validación asegura que estamos desarrollando el producto correcto:

- validación estática revisa la corrección del producto sin ejecutarlo
p.ej., inspecciones, caminatas, y revisiones por pares
- validación dinámica ejecuta el producto o un prototipo
p.ej., testing

Validaciones estáticas: Inspección

Se chequea el software contra una lista de errores, anomalías, y no cumplimiento de estándares y convenciones comunes

Aplicabilidad: Todos los artefactos de software, incluyendo especificación de requisitos, modelo de dominio, casos de uso, diagramas de diseño, código, diseño de la UI, plan de pruebas, casos de prueba

Efectividad: Solo para los problemas comunes incluidos en la lista

Participantes: Uno a tres integrantes del equipo técnico

Procedimiento: Similar a revisión por pares

Errores de programación comunes para inspecciones

variables u otros objetos no inicializados

llamadas a funciones polimórficas incorrectas

llamado incorrecto de funciones

loops que no terminan

desajustes en dimensiones de arreglos

excepciones del tiempo de

ejecución no capturadas

problemas en el uso de punteros

anomalías en el flujo de datos

no reimplementación de features de la clase padre

lógica de negocio incorrecta, inconsistente o incompleta

errores en expresiones aritméticas, relacionales o lógicas

Validaciones estáticas: Caminata

Ejecuta el software manualmente usando datos de prueba simples:

- el desarrollador del software guía al equipo en la caminata; el equipo chequea el software paso a paso a medida que leen en voz alta; la idea es provocar dudas y discusiones

Aplicabilidad: Casos de uso, diagramas de diseño, código, diseño de la UI, casos de prueba

Efectividad: Para entender funcionalidad y comportamiento complejos,

Participantes: Tres a cinco integrantes del equipo técnico, incluyendo el desarrollador

Procedimiento: próxima diapositiva

Procedimiento de caminatas

1. Se presenta una visión global del producto a los participantes
2. El desarrollador lee en voz alta el producto (un documento de requisitos, un módulo de código, etc.) y lo explica;
 - ... los otros participantes preguntan y levantan dudas;
 - ... el desarrollador responde las preguntas y da justificaciones;
 - ... se identifican y se registran las acciones a seguir;
 - ... y se establece un plazo para que el desarrollador corrija los problemas
3. El desarrollador corrige los problemas, produce una lista resumen, y obtiene la aprobación de los participantes

Validaciones estáticas: Revisión por pares

El producto es revisado por pares, que siguen una lista de preguntas de revisión:

- las preguntas evalúan cualitativamente aspectos del producto
- esta evaluación depende del conocimiento, experiencia, etc. del revisor

Aplicabilidad: Todos los artefactos de software

Efectividad: Detección de problemas que requieren el juicio de personas relativos a corrección, eficiencia, amigabilidad, y otros atributos de calidad

Participantes: Tres a cinco revisores con la pericia y experiencia necesarias

Procedimiento: Próxima diapositiva

Procedimiento para inspecciones y revisiones por pares

1. Se presenta una visión global del producto a los revisores; cada unidad del producto es asignada a dos revisores; se planifica una reunión de revisión para dos semanas más
2. Los revisores evalúan el producto contra una checklist y anotan sus respuestas independientemente
3. En la reunión, los revisores presentan sus comentarios y sugerencias y el desarrollador contesta preguntas y clarifica dudas; se identifican y se registran las acciones a seguir; y se establece un plazo para que el desarrollador corrija los problemas
4. El desarrollador corrige los problemas, produce una lista resumen, y obtiene la aprobación de los revisores (o se planifica una segunda reunión, en cuyo caso se repiten los pasos anteriores)

Verificación y validación a lo largo del ciclo de vida

En la fase de requisitos

En la fase de diseño

En la fase de implementación

Funciones de SQA

El objetivo global es asegurar que el proceso de desarrollo de software se lleva a cabo como es requerido

... y que el software satisface los requisitos y estándares de calidad

Abarca una amplia variedad de actividades del ciclo de vida:

- definición de procesos y estándares
- gestión de la calidad
- mejoramiento de procesos

Definición de procesos y estándares

Desarrollar y definir un marco de referencia para asegurar la calidad del software en toda la organización:

- definición de procesos y metodologías de desarrollo de software y gestión de calidad
- definición de estándares, procedimientos y pautas para llevar a cabo las actividades de SQA
- definición de métricas e indicadores de calidad para mediciones y evaluación

Gestión de calidad

Adaptar e implementar el marco de referencia para cada proyecto de software de la organización:

- Planificación de la calidad: Ocurre al comienzo de cada proyecto; produce un plan de calidad para cada proyecto específico
- Control de calidad: Ocurre a lo largo de todo el proyecto; monitorea la ejecución del plan de calidad, y también lo adapta para responder a cambios en la realidad

Mejoramiento de procesos

Es necesario mejorar continuamente el proceso (o los procesos), en todos sus aspectos, para sobrevivir

En particular, los métodos ágiles mejoran de una versión a la siguiente, incluyendo el descubrimiento de mejores formas de adoptar los propios métodos ágiles

Definición y ejecución de un proceso de mejoramiento de procesos (próxima diapositiva), que se lleva a cabo continuamente e itera regularmente; p.ej., cada dos años

El proceso de mejoramiento de procesos

1. Definición de métricas y métodos de recolección de datos
2. Recolección de datos para medir el proceso
3. Calcular métricas e indicadores
4. Recomendar acciones de mejoramiento

Aplicación de principios ágiles

Suficientemente bueno es suficiente

Simple y fácil de hacer

El apoyo de la gerencia es esencial

Un enfoque colaborativo y cooperativo entre todos los interesados es esencial

Valoramos más el software que funciona que la documentación completa

Testing

Es una técnica de validación dinámica

Puede detectar errores en el software,

... pero no puede demostrar que el software no tenga errores

Mejora la confianza del equipo de desarrollo en el producto de software

¿Qué es?

Una disciplina de la ingeniería de software:

- abarca investigación, desarrollo, validación y educación de procesos, métodos y técnicas *cost-effective* para hacer pruebas al software

... y un proceso para asegurar la corrección de un producto de software

... en particular, una actividad de validación dinámica para detectar defectos y comportamientos no deseados en el software:

- errores semánticos, lógicos, de computación
- comportamiento que podría comprometer, p.ej., la seguridad o efectividad del sistema

... y demostrar que el software satisface sus requisitos y restricciones:

- tanto funcionales como no funcionales

¿Por qué es útil?

Los enfoques estáticos de aseguramiento de calidad tienen limitaciones:

- no ejecutan el software, por lo que no pueden detectar errores vinculados a polimorfismo y vinculación dinámica
- es difícil para los revisores humanos lidiar con código complejo

El software es un componente crítico en sistemas de control de procesos, equipamiento médico y aplicaciones de defensa:

- una falla puede causar daños a personas e incluso la pérdida de vidas humanas

Beneficios

Sabiendo que todo software debe ser y va a ser probado, los desarrolladores van a preocuparse más por desarrollar software sin errores

Emplear métodos y herramientas de testing mejorará de manera importante la calidad del software

Testing agrega otra capa —dinámica— de aseguramiento de calidad, además de inspección, caminatas y revisión por pares —estáticas

Si los errores se registran en una base de datos, esta información puede usarse para producir estadísticas e informes de análisis de causas, valiosos para el mejoramiento de procesos

Pruebas de *caja negra*: Pruebas funcionales

P.ej., una función que elimina duplicados de una lista de largo n y produce otra de largo m , $m \leq n$

Caso 1: $n = 0$; $m = n$, el input es la lista vacía

Caso 2: $n = 1$, el input es una lista con un solo elemento

Caso 2.1: $m < n$, error

Caso 2.2: $m = n$, esperado

Caso 3: $n > 1$

Caso 3.1: $m < n$, los duplicados son eliminados

Caso 3.2: $m = n$, la lista de input no tiene elementos duplicados

Especificación de casos de prueba y pruebas producidas a partir de la especificación

Descripción	Input	Output esperado (OE)	Output real (OR)	Criterio para pasar	Resultado
Test lista vacía	lista vacía: ()	igual que input: ()	por determinar	OE = OR	por determinar
Test lista de un elemento	lista con un elemento: (10)	igual que input: (10)	por determinar	OE = OR	por determinar
Test elementos duplicados	lista con duplicados: (1,2,2,3,4,4,5)	lista con duplicados eliminados: (1,2,3,4,5)	por determinar	OE = OR	por determinar
Test sin elementos duplicados	lista sin duplicados: (1,2,3,4,5)	igual que input: (1,2,3,4,5)	por determinar	OE = OR	por determinar

Podría haber más de un prueba en algunos o en todos los casos.
 Además, se podría probar que los elementos en el output estén en el mismo orden que en el input, y que no falte ni sobre ninguno.

Pruebas de *caja negra*: Partición en clases de equivalencia

Divide los dominios del input y del output en subconjuntos disjuntos

... y selecciona un caso de prueba de cada subconjunto:

- hay que identificar una relación de equivalencia —reflexiva, simétrica y transitiva— entre los elementos del dominio del input

El supuesto es que los elementos de una misma partición tienen las mismas características

... por lo que para probar el software con respecto a esas características, es suficiente usar un elemento de cada partición

Pruebas de *caja negra*: Análisis de valores de frontera

Selecciona casos de prueba que están en las fronteras y cerca de las fronteras de las clases de equivalencia (ver próxima diapositiva)

Dominio	Casos de prueba
1) rango de enteros [n1 ... n2]	n1–1, n1, n1+1, n2–1, n2, n2+1
rango de strings consecutivos S1 .. S2	S1, S2, string nulo, string vacío, string muy largo
[n1 ... n2], [n3 ... n4]	Como 1, para cada rango
S1 ... S2, S3 ... S4	S1, S2, S3, S4, string nulo, string vacío, string muy largo
Número único <i>n</i>	n–1, n, n+1
String único <i>S</i>	S, string nulo, string vacío, string muy largo
Enumeración {n1, ..., nk}	nj–1, nj, nj+1, para j = 1, ..., k
Enumeración {S1, ..., Sk}	S1, ..., Sk, string nulo, string vacío, string muy largo
Conjunto de strings	String nulo, string vacío, string muy largo, string de un carácter
{True, False}	{True}, {False}

Pruebas de *caja blanca*: Pruebas de las rutas base

Se generan casos de prueba para ejercitar las rutas de control de flujo independientes (rutas base), a partir del diagrama de flujo del software (ver próxima diapositiva para función **purge**)

Una ruta base es una ruta de B a E y ejercita un ciclo direccional a lo más una vez

Cuatro rutas base para la función **purge**:

B,1,2,E : lista vacía

B,1,2,3,4,9,10,2,E : lista de un elemento

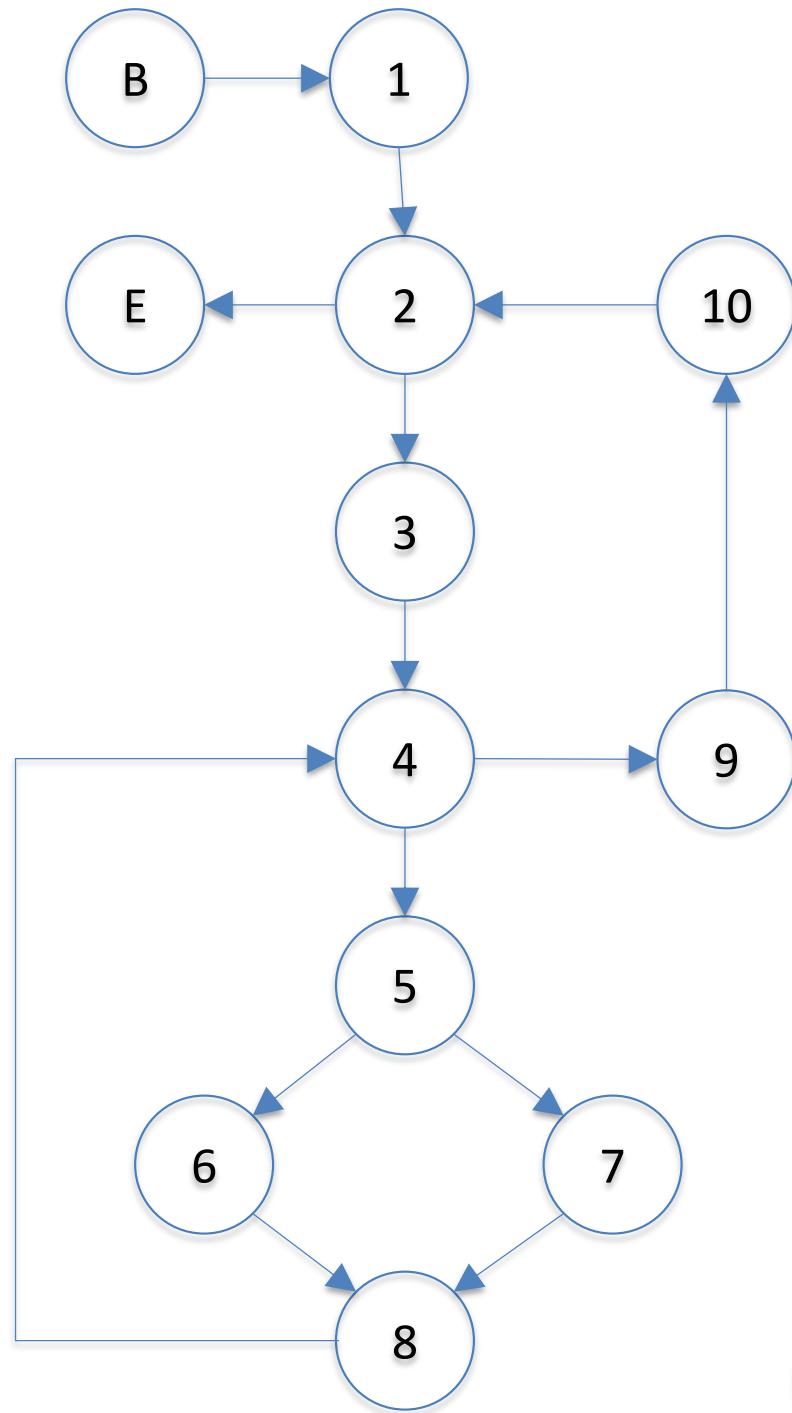
B,1,2,3,4,5,6,8,4,9,10,2,E: lista con duplicados

B,1,2,3,4,5,7,8,4,9,10,2,E: lista sin duplicados

```

purge(L):
1   p = L.first()
2   while p != null:
3       q = p.next()
4       while q != null:
5           if p==q:
6               L.remove(q)
7           else:
8               q = q.next()
9
10  p = p.next()

```



Complejidad ciclomática

Es el número de rutas base del software, calculado de cualquiera de las formas siguientes:

- número de regiones cerradas más uno
- número de aristas menos el número de nodos más dos
- número de condiciones binarias atómicas más uno

Criterios de cobertura de pruebas basados en el diagrama de flujo

Cobertura de nodos: Cada nodo del diagrama debe ser visitado al menos una vez; equivalente a 100% de cobertura de instrucciones

Cobertura de aristas: Cada arista del diagrama debe ser recorrida al menos una vez; equivalente a 100% de cobertura de *branches*

Cobertura de rutas base: Cada ruta base debe ser ejercitada al menos una vez

Cobertura de todas las rutas: Prácticamente imposible, ya que los ciclos anidados pueden crear un número de rutas excesivo

Pruebas de *loops* simples

Saltarse el loop

Una pasada por el loop

Dos pasadas por el loop

Un número típico de iteraciones

Una iteración menos que el número máximo de iteraciones

El número máximo de iteraciones

Intentar una iteración adicional al máximo

Pruebas de *loops* anidados

1. Empezar con el loop más interno; setear todos loops externos en sus valores mínimos
2. Probar el loop más interno:
 - número mínimo de iteraciones, número mínimo de iteraciones más uno, un número típico de iteraciones, número máximo de iteraciones menos uno, número máximo de iteraciones
 - intentar valores fuera de rango: mínimo menos uno, máximo más uno, etc.
3. Avanzar hacia afuera, haciendo pruebas al siguiente loop externo:
 - setear todos los loops externos en sus valores mínimos
 - ejercitar todos los loops anidados con sus valores típicos
4. Repetir los pasos 1 a 3 hasta que todos los loops sean probados

Cobertura de las pruebas

Es el porcentaje de ciertos ítems del software que son probados (o ejercitados), con respecto al número total de ítems del mismo tipo

Es tanto un objetivo de calidad:

- p.ej., requerir 100% de cobertura de *branchs*

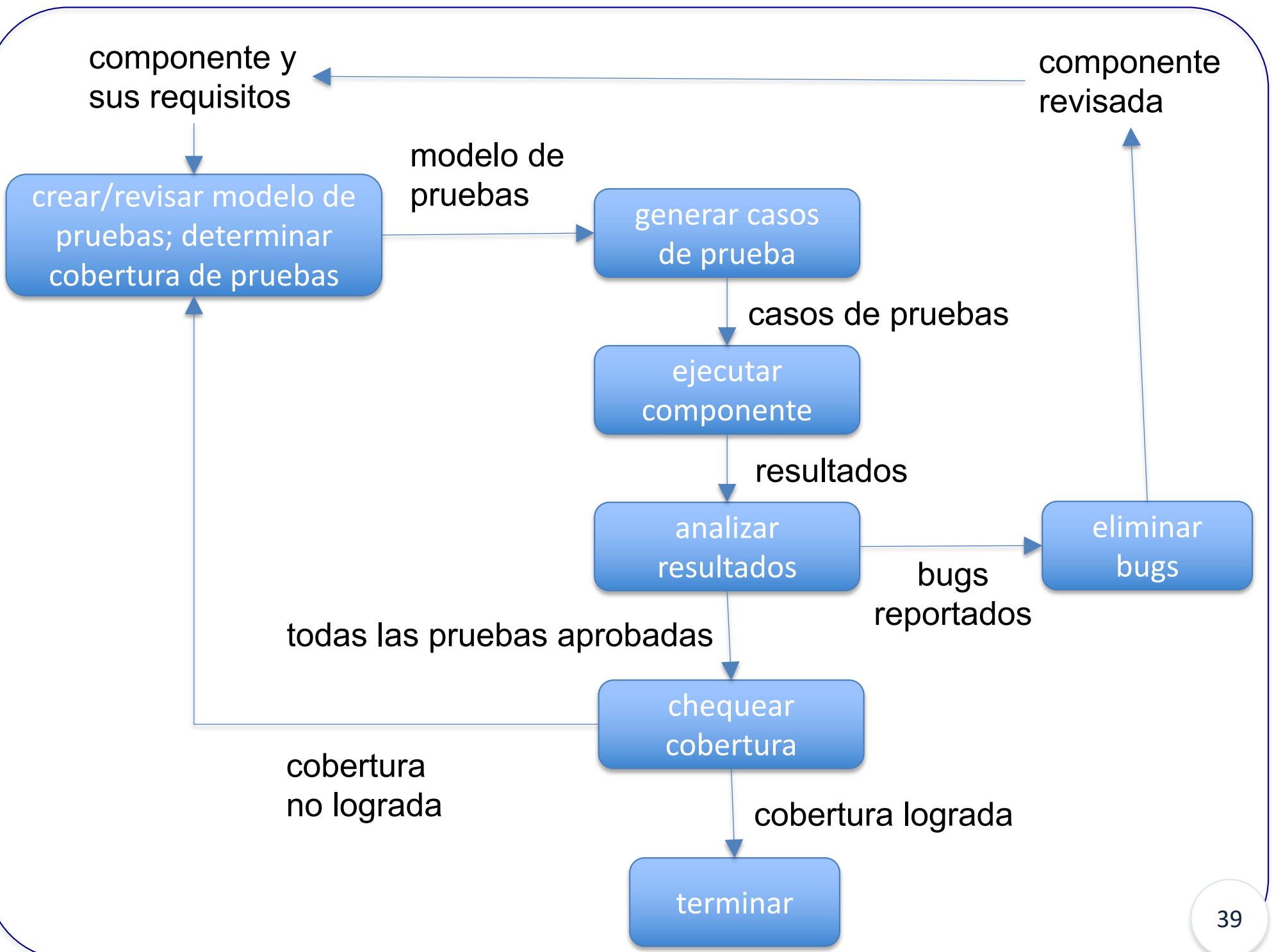
... como una medición del logro de ese objetivo

Un caso práctico es el objetivo de lograr 100% de cobertura de instrucciones, realista para programas de hasta cierto tamaño

No es fácil especificar el criterio de cobertura para pruebas de valores de frontera

Un proceso genérico de pruebas

Próxima diapositiva



Otros tipos de pruebas

Pruebas para software orientado a objetos:

- ... basadas en casos de uso
- ... basadas en el estado del objeto
- ... de la jerarquía de clases
- ... del manejo de excepciones

Pruebas para requisitos no funcionales:

- ... de desempeño y *stress*
- ... de seguridad
- ... de la interfaz de usuario

Otros temas:

- Pruebas a lo largo del ciclo de vida
- Pruebas de regresión
- ¿Cuándo terminar de hacer pruebas?

Aplicación de principios ágiles

Hacer testing a través de todo el ciclo de vida, desde temprano y a menudo

Practicar el desarrollo dirigido por pruebas (TDD)