Testen von Software II

Software Engineering II (IB)

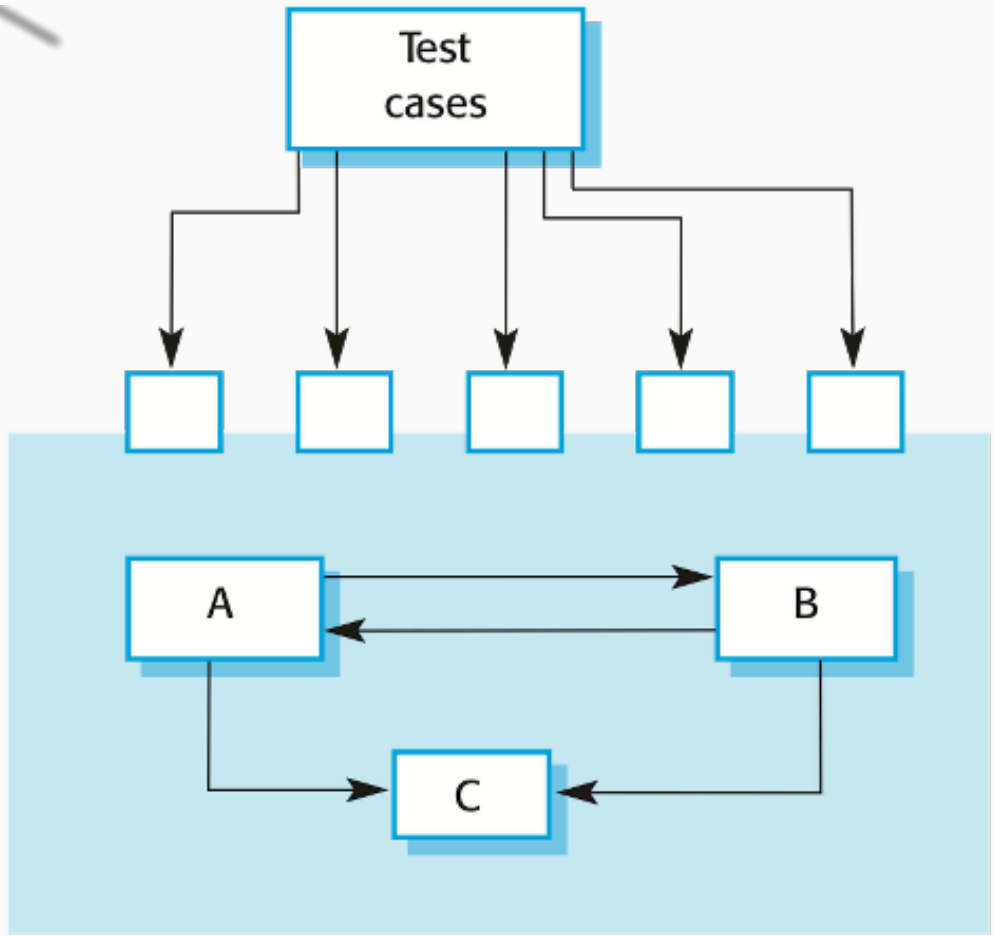
Prof. Dr. Oliver Braun

Letzte Änderung: 07.04.2019 10:40

Testen von Komponenten

- Voraussetzung: Modultests für die einzelnen Objekte innerhalb der Komponente abgeschlossen
- Funktionalität der Objekte wird über das definierte Komponenteninterface getestet
- Ziel ist zu testen, ob sich das Interface passend zur Spezifikation verhält

Schnittstellentests



Schnittstellentests (Quelle: I. Sommerville: Software Engineering)

Arten von Schnittstellen

Parameterschnittstellen Daten oder Funktionsreferenzen werden übergeben, z.B. Methoden

Schnittstellen mit gemeinsamem Speicher häufig in eingebetteten Systemen

Prozedurschnittstellen Subsystem kapselt eine Reihe von Prozeduren

Schnittstellen zur Nachrichtenübergabe z.B. in Client-Server-Systemen

Klassen von Schnittstellenfehlern

1. Falsche Verwendung der Schnittstelle
    - z.B. Aufruf mit falscher Anzahl Parameter
2. Schnittstellenmissverständnis
    - aufrufende Komponente versteht die Schnittstelle falsch und setzt ein bestimmtes Verhalten der aufgerufenen Komponente voraus
    - z.B. Aufruf binäre Suchmethode mit einem ungeordneten Feld
3. Zeitabstimmungsfehler
    - in Echtzeitsystemen
    - Produzent und Konsument arbeiten möglicherweise mit unterschiedlicher Geschwindigkeit

Richtlinien für Schnittstellentests

- entwerfen Sie Tests so, dass die Parameter an den äußeren Enden ihrer Gültigkeitsbereiche liegen
- testen Sie Pointer mit Null-Pointern
- entwerfen Sie Tests die absichtlich zum Versagen der Komponente führen
- verwenden Sie bei Nachrichtenaustauschsystemen Lasttests
- wenn verschiedene Komponenten über gemeinsamen Speicher zusammenarbeiten, entwerfen Sie Tests, die die Reihenfolge der Aktivierung der Komponenten variieren

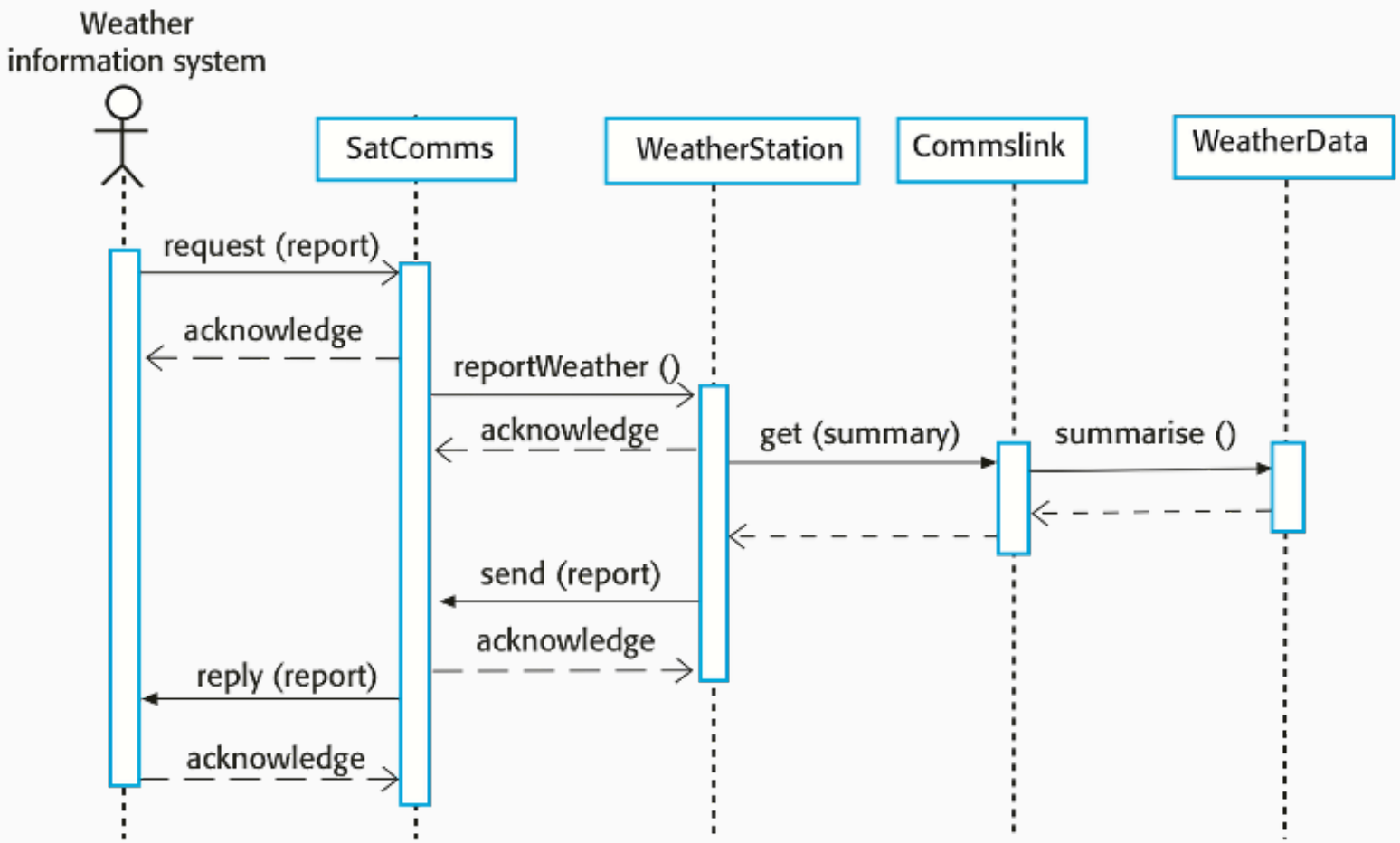
Testen von Systemen

- beim Testen des Systems während der Entwicklung werden die Komponenten integriert, die eine Version erzeugen und das integrierte System getestet
- Systemtests überprüfen ob die Komponenten miteinander kompatibel sind und korrekt miteinander interagieren
- Systemtests Testen das emergente Verhalten, d.h. ein Verhalten, das erst durch Zusammenfügen der Komponenten sichtbar wird
    - z.B. Authentifizierungskomponente und andere Komponente bieten gemeinsam erst die Möglichkeit, dass bestimmte Funktionen auf bestimmte Nutzer beschränkt sind

Anwendungsfalltesten

- die Use-Cases mit denen System-Interaktionen identifiziert wurden, sind eine gute Ausgangsbasis für Systemtests
- jeder Use-Case beinhaltet normalerweise mehrere Systemkomponenten
- die Sequenzdiagramme die mit den Use-Cases assoziert sind, dokumentieren die Komponenten und die Interaktionen die getestet werden müssen

Sequenzdiagramm zur Sammlung von Wetterdaten



Sequenzdiagramm zur Sammlung von Wetterdaten (Quelle: I. Sommerville: Software Engineering)

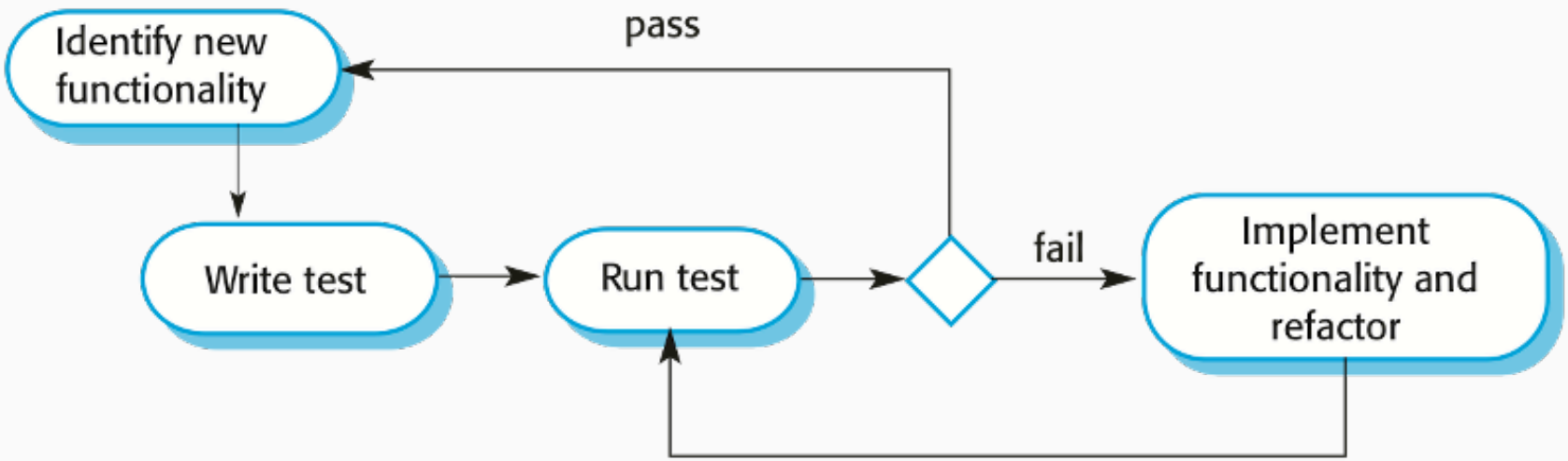
Richtlinien

- vollständiges Systemtesten ist unmöglich
- daher sind Richtlinien wichtig
- Beispiele für Richtlinien
  - alle über Menüs erreichbaren Systemfunktionen sollten getestet werden
  - Kombinationen von Funktionen (z.B. Textformatierungen), auf die über das selbe Menü zugegriffen wird, müssen getestet werden
  - wenn Benutzereingaben eine Rolle spielen, müssen alle Funktionen sowohl mit richtigen als auch mit falschen Eingaben getestet werden

Testgetriebene Entwicklung

- Test-driven development (TDD) ist ein Ansatz zur Programmentwicklung bei der Testen und Codeentwicklung ineinander greifen
- Tests werden vor dem Code geschrieben und das “Erfüllen” des Tests ist der Antrieb bei der Entwicklung
- Code wird inkrementell zusammen mit einem Test für dieses Inkrement entwickelt
- es wird erst dann zum nächsten Inkrement übergegangen, wenn der Test “grün” ist
- TDD kommt aus der agilen Entwicklung

TDD



TDD (Quelle: I. Sommerville: Software Engineering)

TDD-Prozesse

1. Beginnen Sie mit der Identifizierung der geforderten Funktionalität. Normalerweise klein und in wenigen Codezeilen implementierbar.
2. Schreiben Sie einen automatisierbaren Test dafür.
3. Führen Sie den Test zusammen mit allen bisherigen aus.
4. Implementieren Sie die Funktionalität und wiederholen Sie den Test.
5. Wenn alle Tests erfolgreich sind, gehen Sie zum nächsten Funktionalitätsbaustein über.

Vorteile von TDD

Codeabdeckung für jedes Codesegment gibt es mindestens einen Test

Regressionstests die Testsuite wird inkrementell entwickelt

Vereinfachtes Fehlerbeheben wenn ein Test fehlschlägt, ist es offensichtlich wo das Problem ist

Systemdokumentation die Tests sind eine Art Dokumentation

Regressionstests

- Regressionstests testen ob Änderung an einem System nichts zerstört haben was zuvor lief
- beim manuellen Testen ist das sehr aufwändig
- durch automatisierte Tests ist das billig
- Tests müssen erfolgreich sein, bevor Änderungen committed werden

Freigabetests

- Release Testing testet eine Version des Systems die außerhalb des Entwicklungsteams genutzt werden soll
- primäres Ziel ist es zu zeigen, dass das System gut genug ist
    - spezifizierte Funktionalität
    - Performanz
    - Zuverlässigkeit
- Blackbox-Test mit aus der Systemspezifikation abgeleiteten Testfällen

Freigabetests vs. Systemtests

- Freigabetests sind eine Form von Systemtests
- wichtige Unterschiede
  - Freigabetests sollten von einem separaten Team, das nicht an der Entwicklung beteiligt war, durchgeführt werden.
  - Systemtests von den Entwicklern sollten Fehlertests sein, Freigabetests hingegen Validierungstests

Anforderungsbasiertes Testen

- gute Anforderungen sollten testbar sein
- requirements based testing ist eine systematische Herangehensweise an den Testentwurf
- jede Anforderung wird in Betracht gezogen und eine Testreihe für sie gestaltet
- eher Validierungstests

Szenariobasiertes Testen

- typische Benutzerszenarios finden und dazu Testfälle entwicklen
- ein Szenario ist eine Geschichte, die eine Möglichkeit beschreibt, das System zu nutzen
- Beispiele
  - Authentfizierung bei der Anmeldung im System
  - Reservieren und Ausleihen eines Skis
  - ...

Leistungstests

- Teil der Freigabetests sind Leistungstests
- bei Performancetests wird normalerweise eine Testserie ausgeführt, bei der die Belastung solange erhöht wird, bis die Systemleistung inakzeptabel wird
- Lasttests (Stresstests) sind eine Form von Leistungstest bei denen Forderungen gestellt werden, die außerhalb der vorgegebenen Grenzen der Software liegen

Benutzertests

- Benutzer bzw. Kunden testen das System
- Arten

Alphatests Benutzer arbeiten mit dem Entwicklungsteam zusammen um die Software in der Entwicklungsumgebung zu testen.

Betatests Benutzern wird ein Release der Software zur Verfügung gestellt, mit dem sie experimentieren können und die dabei entdeckten Probleme mit den Systementwicklern besprechen können.

Abnahmetests (acceptance tests) Kunden testen ein System um zu entscheiden ob sie es abnehmen und in ihrer Umgebung installieren können.

Phasen im Abnahmetest

1. Abnahmekriterien festlegen
2. Abnahmetests planen
3. Abnahmetests ableiten
4. Abnahmetests ausführen
5. Testergebnisse verhandeln
6. das System ablehnen bzw. akzeptieren

Agile Methoden und Abnahmetests

- Benutzer ist Teil des Entwicklungsteams (Alphatester)
- Tests werden durch den Benutzer definiert und mit den anderen automatisch ausgeführt
- kein separater Abnahmetestprozess
- Problem ist nur ob der Benutzer “typisch” ist und alle Interessen aller Stakeholder repräsentiert