8. Test und Integration

Warum Testen?

Die Top 5 der Software-Katastrophen:

* **Familienkasse verteilt Geschenke**  
  Die britische Familienbehörde (CSA) führte 2004 ein neues IT-System ein, das dazu führte, dass 1.9 Millionen Menschen zuviel Geld ausbezahlt bekamen, 700.000 andere zu wenig. 7 Milliarden Dollar Steuern wurden zu wenig bezahlt, 240.000 Fälle wurden nicht bearbeitet.  
  Grund: Zur gleichen Zeit der Einführung des Systems beschloss das Arbeitsministerium, die Familienbehörde umzustrukturieren. Die Software passte nicht mehr zur neuen Struktur und wurde umgeschrieben – wobei schwere Fehler ins System eingeschrieben wurden, 500 bekannte Bugs wurden nie behoben.
* **Mariner 1 Crash**  
  1962 brach die Mariner-1-Rakete zu einer Mission zur Venus auf – und triftete kurz nach dem Start von ihrer geplanten Laufbahn ab. Die Mariner 1 musste zerstört werden, über 18 Millionen US-Dollar wurden in den Sand gesetzt.  
  Grund: Ein Programmierer übertrug eine Formel nicht korrekt von einem handgeschriebenen Dokument in den Quellcode.
* **22 Verhaftungen wegen fehlerhafter Datenbankmigration**   
  In einer australischen Polizei-Datenbank wurden 2011 beim Datentransfer 3.600 Datensätze fehlerhaft übertragen. Resultat waren 22 [Verhaftungen unschuldiger Personen](http://www.brisbanetimes.com.au/technology/technology-news/police-force-damaged-by-computer-errors-20110608-1ft9n.html).
* **War Games**   
  Das Frühwarnsystem der sowjetischen Streitkräfte meldete 1983 fünf ballistische Raketen, die von US-Territorium auf die UdSSR abgeschossen wurden. Glücklicherweise interpretierte ein Offizier diese Meldung als Fehlalarm – wenn die USA tatsächlich angreifen würden, hätten sie mehr als fünf Raketen abgeschossen.  
  Grund: Ein Bug in der sowjetischen Früherkennungs-Software verhinderte die Unterscheidung zwischen Raketen und Sonnenreflektionen auf Wolken.
* **Deutsches Maut-Desaster**  
  2003 versuchte das deutsche Toll Collect die politisch beschlossene LKW-Maut einzuführen. Nachdem ausgelieferte Hardware für identische Strecken unterschiedliche Gebühren anzeigte, wurde eine Rückrufaktion eingeleitet. Obwohl bereits für 2003 geplant, wurde der volle Betrieb des Systems erst 2006 möglich. 3,5 Milliarden Euro Einnahmeausfälle wurden in einem Verfahren gegen das Unternehmen geltend gemacht. Bis heute streitet man sich über das Verhältnis zwischen politischen, technischen und organisatorischen Fehlern.

Testen im Allgemeinen:

* Es gibt Blackbox-Tests und Whitebox-Tests.
* Soll prüfen ob die SW tut was sie tun soll, bevor die SW in den Einsatz kommt
* Man gibt dem Test künstliche Daten zum prüfen
* Fehler kann man nachweisen, aber nicht die Abwesenheit von Fehlern.
* Test ist Teil eines Validierungsprozesses, welcher auch statische Techniken umfasst.

Ziele:

* Dem Entwickler und dem Kunden zeigen, dass die Software die an sie gestellte Anforderungen erfüllt. 🡪 Validierungsetests
* Situation aufzuspüren, in denen sich die Software falsch, unerwünscht oder nicht spezifikationsgemäß verhält. 🡪 Fehlertests

Vertifikation und Validierung

Diese Unterscheidung spielt bei den Vorgehensmodellen (siehe später) wieder eine Rolle.

1. **Verifikation:** Bauen wir das Produkt richtig?Prüfen gegen die Spezifikation

**Validierung**: Bauen wir das richtige Produkt?Prüfen gegen die wirklichen Bedürfnisse des Nutzers.

Der Verifikations- und Validierungs-Prozess schafft das Vertrauen, dass die Software ihren Zweck erfüllt. Wie tief das Vertrauen gehen muss, hängt von einigen Faktoren ab:

* Zweck der Software
* Erwartungen der Benutzer
* Marktsituation

Neben Tests (dynamische Tests) spielen im V&V Prozess (Vertifikation und Validierung) auch statische Tests eine Rolle.

Vorteile von Software Inspektionen:

Beim Testen können Fehler anderer Fehler überdecken.

Man kann unvollständige Versionen untersuchen aber nicht ohne weiteres testen.

Man kann über die Fehlersuche hinaus Qualitätsmerkmale prüfen.

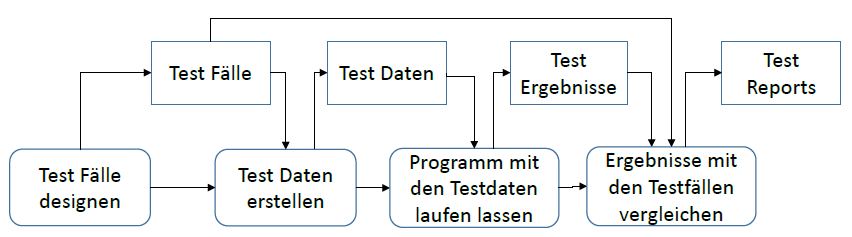
Nachteile von SW Inspektionen:

Inspektionen können nicht oder nur schwer die nicht-fkt Charakteristiken überprüfen.

Inspektionen können prüfen, ob Spezifikationen eingehalten werden, aber nicht, ob die wirklichen Benutzeranforderungen erfüllt sind.

Inspektionen und Tests ergänzen sich.

Abstraktes Modell des traditionellen Testprozesses:



Es gibt drei Testphasen: Entwicklertests, Freigabetests, Benutzertests

1. **Entwicklerstests**:

Entwicklertests umfassen alle Testaktivitäten, die vom Entwicklerteam während der Entwicklung durchgeführt werden.

Es gibt drei Unterstufen:

* Modultests (unit Testing) 🡪 Objekte und Methoden
* Komponententests 🡪 Komponentenschnittstellen
* Systemtests 🡪 Interaktionen zwischen Komponenten

Modeltests - Unit Testing:

Hier testet man einzelne Einheiten isoliert. Diese Einheiten sind typischerweise einzelne Methoden, Klassen mit mehreren Attributen und Methoden, zusammengesetzte Komponenten mit definierten Schnittstellen.

Ziel dabei ist es, alle Eigenschaften des Objekts abzudecken. Das heißt alle Operationen der Klasse, alle Attributwerte, alle mögl. Zustände.

Die Testfälle sollen zeigen, dass die Komponete tut was sie tun soll. Wenn Defekte da sind, soll man diese aufzeigen.

Es gibt zwei Arten von Unit Test Cases:

Die normale Programmausführung (das ding tut was es tun soll). Und die Ungewöhnliche Eingabe, falsche Eingaben etc. Die Komponenten kann damit umgehen, ohne zu crashen.

Automatisierte Modultests:

Auswahl der Testfälle:

Dabei gibt es zwei Strategien:

1. **(Äquivalenz-)Klassenbasierte Tests**: Identifikation von Gruppen von Eingabedaten mit ähnlichem Verhalten 🡪 mind. ein Test pro Gruppe.
2. **Richtlinienbasierte Tests**: Verwendung von Richtlinien, um die Testfälle zu definieren.

Äquivalenzklassentest:

**S.22**