

Software Engineering

Software Test

Prof. Dr. Peter Jüttner

Hochschule Deggendorf

Literatur



Andreas Spillner, Tilo Linz Basiswissen Softwaretest
<http://www.dpunkt.de/certified-tester/> German Book



Beizer, Boris Black-Box Testing: Techniques for Functional Testing of Software and Systems



Peter Liggesmeyer Software-Qualität: Testen, Analysieren und Verifizieren von Software



Harry M. Sneed, Mario Winter Testen objektorientierter Software

Inhalte

5. Methoden

5.1. Requirments Engineering

5.2 Architektur & Feindesign

5.3 Implementierung

5.4 Test

5.5 Qualitätssicherung

5.6 Projektmanagement

5.7 Konfigurationsmanagement

6. Der Faktor Mensch in der SW Entwicklung

Inhalte

5.4 Test

5.4.1 Statischer Test

5.4.2 Dynamischer Test

5.4.3 Testphasen

5.4.3.1 Modultest

5.4.3.2 Integrationstest

5.4.3.3 Validierung

5.4.4. Abgrenzung SW-Test <-> Systemtest

5.4.5. Testtechniken und Teststrategien

5.4.5.1. Black-Box

5.4.5.2. White-Box

Inhalte

- 5.4.6 Weitere Begriffe – Regressions- und Abnahmetest
- 5.4.7 Test Dokumentation
- 5.4.8 Exkurs: Testen im Automotive Umfeld
- 5.4.9 Reduktion des Testumfangs
- 5.4.10 Exkurs: Testen objektorientierter Software
- 5.4.11 Exkurs: Testdatengenerierung aus UML Modellen
- 5.4.12 Exkurs: Modellbasiertes Testen
- 5.4.13 Exkurs: Testen grafischer Bedienoberflächen
- 5.4.14 Do's und Dont's, Erfahrungen, Probleme

Übung

- **Was ist das Ziel der SW Tests?**
- **Was wird zum SW Test benötigt?**

5 min, arbeiten Sie ggf. zusammen mit einem Partner



Definition / Was ist Software Testen?

Software Testen ist

"The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software items"

[IEEE 829 Definition].

Motivation

Software Testen - Warum

- Testen ist **die** Methode, Fehler in einer Software zu entdecken
 - Qualitätsanforderungen des Kunden
 - Produkthaftung
 - Sicherheitskritische Systeme (z.B. Kraftwerke, Flugzeuge, Fahrzeuge, medizinische Geräte)
- ⇒ **Vertrauen schaffen in die zu testende Software**

Fehlerbehebungskosten

~ 1 k€	~ 1.5 k€	~ 3 k €	~ 6 k €	~ 12 k €	>250 € pro Fzg.!	Kosten einer Fehlerkorrektur
Require- ments	Design	Code	SW Test	System Test	Feld	
Fehlervermeidung			Fehlerfinden			

Verringerung der Fehlerkosten:

1. Fehlervermeidung
2. Je früher desto günstiger.
3. Je früher ein Fehler gefunden wird, desto höher die Wahrscheinlichkeit, dass durch die Korrektur keine neuen Fehler entstehen.

Source: SV I IP internal data, 2002

Gesetzliche Anforderungen - Produkthaftung



Gesetz über die Haftung für fehlerhafte Produkte (Produkthaftungsgesetz - ProdHaftG)

1. Haftung

(1) Wird durch den Fehler eines Produkts jemand getötet, sein Körper oder seine Gesundheit verletzt oder eine Sache beschädigt, so ist der Hersteller des Produkts verpflichtet, dem Geschädigten den daraus entstehenden Schaden zu ersetzen. [..]

(2) Die Ersatzpflicht des Herstellers ist ausgeschlossen, wenn

...

5. der Fehler nach dem Stand der Wissenschaft und Technik in dem Zeitpunkt, in dem der Hersteller das Produkt in den Verkehr brachte, nicht erkannt werden konnte.

[..]

Abgrenzung

Testen ist ...

- **Fehler finden**
- **Vertrauen in die Qualität der SW schaffen**
- Stichprobe, d.h. es gibt eine Wahrscheinlichkeit, dass beim Testen ein Fehler auftritt
- Immer anwendbar
- Planbar (Aufwand, Dauer)
- Spezialisierte Tester (als Rolle im SW Engineering)



Abgrenzung

Testen ist nicht ...

- **Der Beweis (mathematischer) Korrektheit**
 - Verfahren für Korrektheitsbeweise existieren für bestimmte Anwendungsfälle (z.B. Zustandsautomaten)

Mathematische Beweise sind leider nicht allgemein anwendbar



Abgrenzung

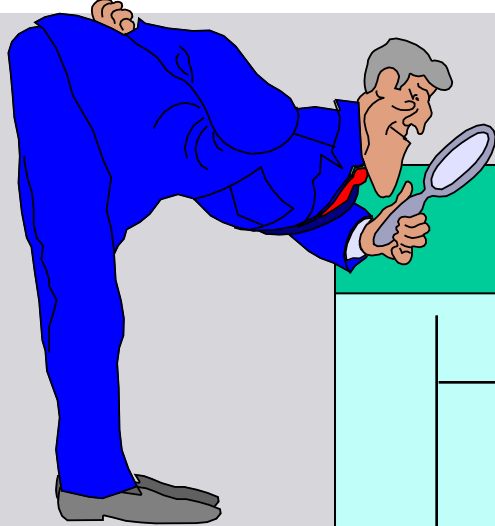
Testen ist nicht ...

- **Debuggen**

- Existenz eines Fehlers bereits bekannt
- Ziel: Fehlerursache und Fehlerort finden
- Basiert auf Design und Code
- Planung manchmal schwierig
- Meist durch Entwickler durchgeführt



Definitionen



Software Test

Statischer Test (*Keine Ausführung von Code*)

- Review**

- Statische Prüfung (mittels Tools)**

Dynamischer Test

- Ausführung von Code**

- Ausführbare Spezifikationen und Modelle**

- Simulationen**

Definitionen

- Review
- ➔ wird im Kapitel Qualitätssicherung besprochen

Motivation

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4 Test

5.4.1 Statischer Test

5.4.2 Dynamischer Test

5.4.3 Testphasen

5.4.3.1 Modultest

5.4.3.2 Integrationstest

5.4.3.3 Validierung

...

Definitionen

- Statischer Test
 - Kein Ausführen des Programms
 - Benötigt daher keine Testfälle
 - Visuell durch Reviews
 - Reviews ggf. unterstützt durch Checklisten
 - Automatisch durch Tools

Definitionen

- Statischer Test
 - Ziele u.a.
 - Finden von „echten“ Fehlern (z.B. nicht initialisierte Variable, statische Arraygrenzenverletzung)
 - Finden von potentiellen Schwachstellen (z.B. Kombination von signed/unsigned Variablen, Zuweisung von Variablen unterschiedlicher Längen)
 - Einhaltung von Programmierstandards (z.B. MISRA)
 - Einhaltung von Programmierkonventionen (z.B. Namensgebung von Variablen und Funktionen)
 - Sicherstellen von Portierbarkeit
 - Sicherstellen von Wartbarkeit
 - Sicherstellen von Verständlichkeit

Definitionen

- Statischer Test – Tools (Beispiele)
 - PC Lint
 - Statischer Codechecker für C/C++
 - Gimpel Software, www.gimpel.com
 - MISRA (teilweise)
 - QAC
 - Statischer Codechecker
 - Metriken
 - Konfigurierbar bzgl. eigener Regeln
 - QA-Systems, www.qa-systems.de
 - MISRA

Definitionen

PC-Lint Beispielergebnisse:

- f.cpp (9) : Info 1732: new in constructor for class 'X' which has no assignment operator x.cpp (9) : Info 1733: new in constructor for class 'X' which has no copy constructor
- f.cpp (21) : Warning 613: Possible use of null pointer 'p' in argument to operator '++' [Reference: file f.cpp: line 20]
- f.cpp (21) : Info 810: Arithmetic modification of custodial pointer 'p'
- f.cpp (25) : Warning 669: Possible data overrun for function 'strcpy(char *, const char *)', argument 2 (size=100) exceeds argument 1 (size=99) [Reference: file f.cpp: lines 20, 21, 24, 25]

Motivation

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4 Test

5.4.1 Statischer Test

5.4.2 Dynamischer Test

5.4.3 Testphasen

5.4.3.1 Modultest

5.4.3.2 Integrationstest

5.4.3.3 Validierung

5.4.4. Abgrenzung SW-Test <-> Systemtest

5.4.5. Testtechniken und Teststrategien

...

Definitionen

- Dynamischer Test
 - Ausführen der zu testenden Software
 - Ausführen von Spezifikationen oder Modellen
 - Benötigt Testfälle
 - Ziele u.a.
 - Finden von Fehlern in Algorithmen
 - Finden von Fehlern im Zusammenspiel verschiedener Programmteile
 - Finden von Fehlern im dynamischen Verhalten (Laufzeit, Speicherverbrauch)
 - Sicherstellen von Robustheit
 - Sicherstellen der Funktion über einen längeren Zeitraum
 - Sicherstellen eines Wiederanlaufs

Definitionen

- Statischer und Dynamischer Test
 - Ergänzen einander
 - Sind bzgl. Fehlerfindung teilweise disjunkt
 - D.h. sollte immer kombiniert durchgeführt werden
 - Codecheck durch Entwickler
 - Review durch andere Entwickler
 - Dynamischer Test durch (wenn möglich **unabhängige(!)** Tester, d.h. Designer / Codierer \neq Tester)

Definitionen

Testfall

Ein Testfall beschreibt einen **elementaren, funktionalen** Softwaretest, der der Überprüfung einer z.B. in einer Spezifikation zugesicherten Eigenschaft eines Testobjektes dient.

Ein Testfall wird mittels Testmethoden (s. Teststrategie, Testtechnik) erstellt.

Bezug Testfall \leftrightarrow Anforderung : **Nachverfolgbarkeit
(Traceability)**

Definitionen

Wichtige Bestandteile eines Testfalls sind:

- die Vorbedingungen für die Testausführung
- die Eingaben/Handlungen, die zur Durchführung des Testfalls notwendig sind,
- ggf. Informationen, wie ein verwendetes Testtool zu konfigurieren oder zu steuern ist,
- die erwarteten Ausgaben/Reaktionen des Testobjektes auf die Eingaben,
- die erwarteten Nachbedingungen, die als Ergebnis der Durchführung des Testfalls erzielt werden.

Definitionen

Wichtige Bestandteile eines Testfalls sind:

- die Prüfanweisungen, d.h. wie Eingaben an das Testobjekt zu übergeben sind und wie Sollwerte abzulesen sind
- Weichen die Ausgaben oder Nachbedingungen während des Testlaufs/der Testdurchführung von den erwarteten Werten ab, so liegt eine Anomalie vor (das kann z.B. ein zu behebender Mangel oder Fehler sein, kann aber auch ein Fehler im Testfall sein).
- Dokumentation des Testfalls
 - Beschreibung des Testfalls
 - Festhalten des Ergebnis

Definitionen

Beispiel Requirement und zugehöriger Testfall:

Req.107 (Steuergerät): Falls die Spannung für mindestens 5 Sekunden über 14V steigt, soll ein Überspannungsfehler eingetragen werden.

Testfall:

- Fehlerspeicher gelöscht (Vorbedingung)
- Steuergerät an Spannungsversorgung 12V angeschlossen (Vorbedingung)
- Spannung für 5 Sekunden auf 14,1V einstellen (Eingabe/Handlung)
- Fehlerspeicher auslesen (Handlung)
- Überspannungsfehler ist eingetragen (Sollwert)

Motivation

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4 Test

5.4.1 Statischer Test

5.4.2 Dynamischer Test

5.4.3 Testphasen

5.4.3.1 Modultest

5.4.3.2 Integrationstest

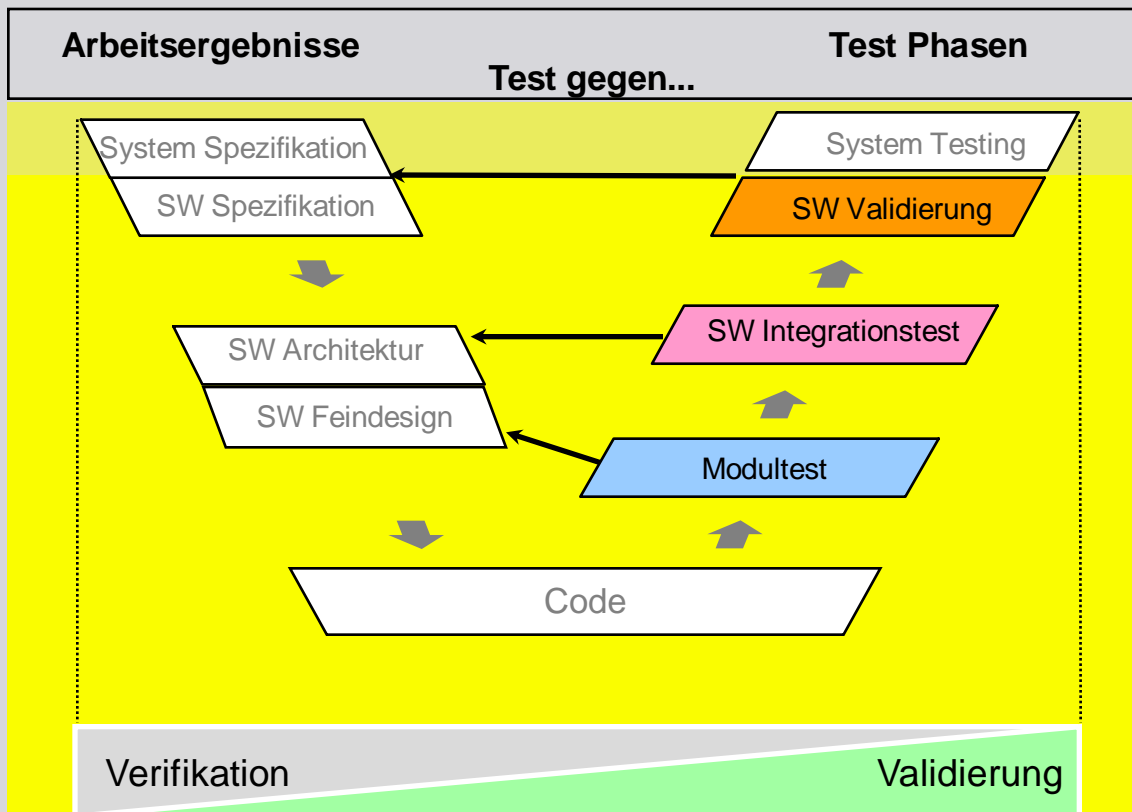
5.4.3.3 Validierung

5.4.4. Abgrenzung SW-Test <-> Systemtest

5.4.5. Testtechniken und Teststrategien

...

Test Phasen – Prozess, Verifikation, Validierung



Test Prozess:

Sequenz von Schritten, die ausgeführt werden um eine Software zu testen. Diese Schritte werden in drei **Test Phasen** angeordnet.

Verifikation:

Test gegen die Ergebnisse der vorhergehenden Phase.

Validierung:

Test gegen die Spezifikation bzw. gegen die Erwartungen des Anwenders.

Test Phasen – Prozess, Verifikation, Validierung



Vereinfacht gesagt ...

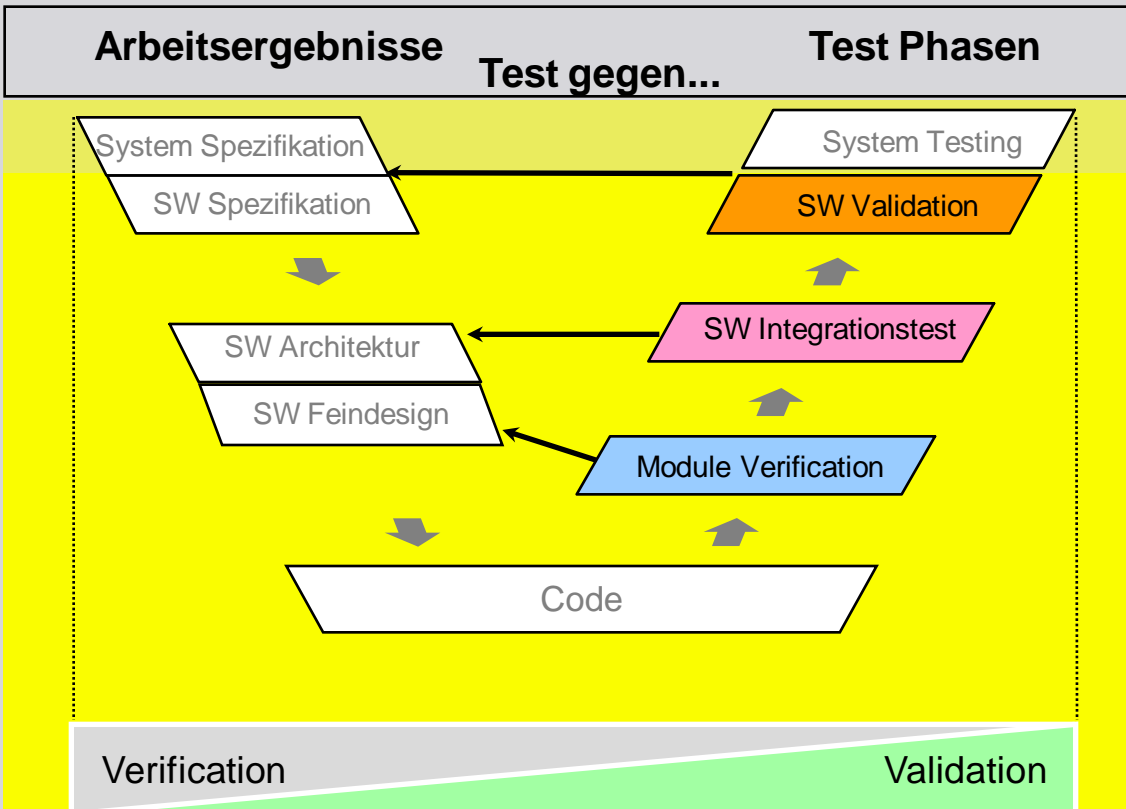
- Verifikation

Sicherstellen, dass das Produkt **richtig entwickelt** wurde

- Validierung

Sicherstellen, dass das **richtige Produkt** entwickelt wurde

Test Phasen – Test Prozess



Phasen:

Modultest:

Test einer individuellen Komponente.
Synonym **Unittest**.

SW Integrationstest:

Zusammenfügen und Testen bereits (modul-)getester SW-Komponenten.
Literatur **SW integration**

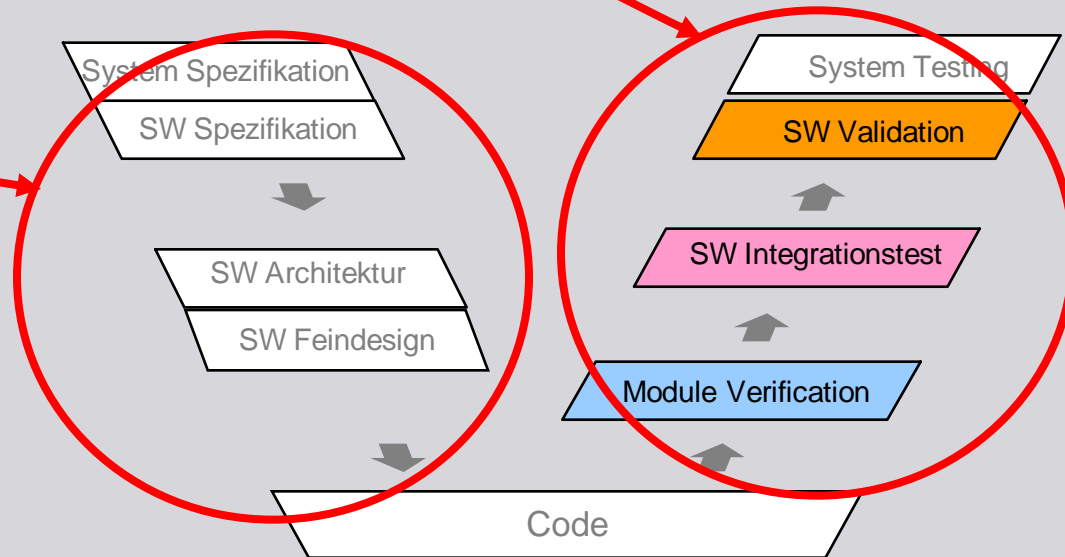
SW Validation:

Demonstrieren, dass die Software Anforderungen nicht erfüllt.

Test Phasen – Test Prozess

Achtung:

- Die Test Phasen beziehen sich nur (!) auf die Ausführung bzw. Auswertung des Tests („rechter Teil des „V““)
- Die Spezifikation und Implementierung der Tests werden während Spezifikation, Design und Codierung durchgeführt („linker Teil des „V““)
- Tests werden während des kompletten „V“s gemanagt (z.B. Planung, Verfolgung, Konfigurationsmanagement)



Motivation

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4 Test

5.4.1 Statischer Test

5.4.2 Dynamischer Test

5.4.3 Testphasen

5.4.3.1 Modultest

5.4.3.2 Integrationstest

5.4.3.3 Validierung

5.4.4. Abgrenzung SW-Test <-> Systemtest

5.4.5. Testtechniken und Teststrategien

...

Test Phasen – Modultest

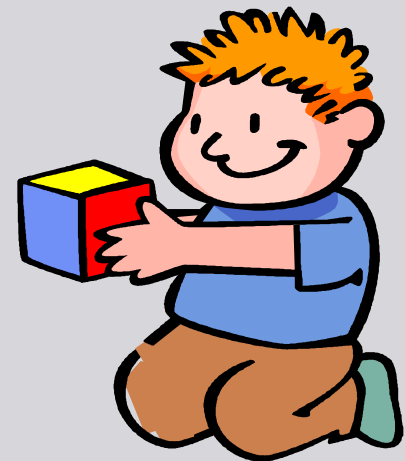
Test einer individuellen SW Komponente (e.g. C-Funktion, Subroutine), wobei die Komponente u.U. in weitere Unterkomponenten zerlegt werden kann.

Die Komponente wird in Isolation (!) getestet, d.h. die Umgebung wird ggf. simuliert.

(Synonym: Unittest)

Ziel:

- lokale Fehler finden,
- Abweichungen vom (Fein-) Design finden

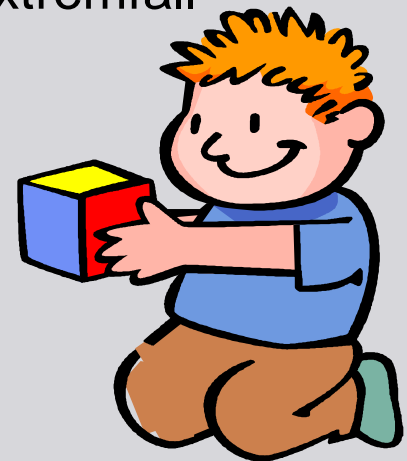


Test Phasen – Modultest

Beim Modultest werden

- Testtreiber benötigt, die das zu testende Modul aufrufen
- Simulationen der Umgebung (z.B. aufgerufene Module) durch Stubs (Dummies) durchgeführt. Ein Stub ersetzt das Original nur für die durchgeführten Testfälle. Stubs müssen u.U. auch getestet werden.

Anmerkung: Stubs können sehr aufwendig werden. Im Extremfall so aufwendig, wie das zu ersetzende Modul



Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4 Test

5.4.1 Statischer Test

5.4.2 Dynamischer Test

5.4.3 Testphasen

5.4.3.1 Modultest

5.4.3.2 Integrationstest

5.4.3.3 Validierung

5.4.4. Abgrenzung SW-Test <-> Systemtest

5.4.5. Testtechniken und Teststrategien

...

Test Phasen – Integrationstest

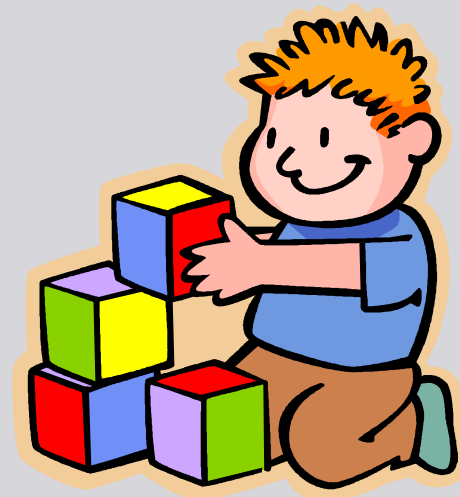
Der Prozess des Zusammenbauens und Testens bereits getesteter (!) Module.

Die SW Architektur bildet die Basis der SW Integration , d.h.

Die SW Integration ist ein Test gegen die SW Architektur, d.h. bzgl. interner Schnittstellen und Abläufe z.B.

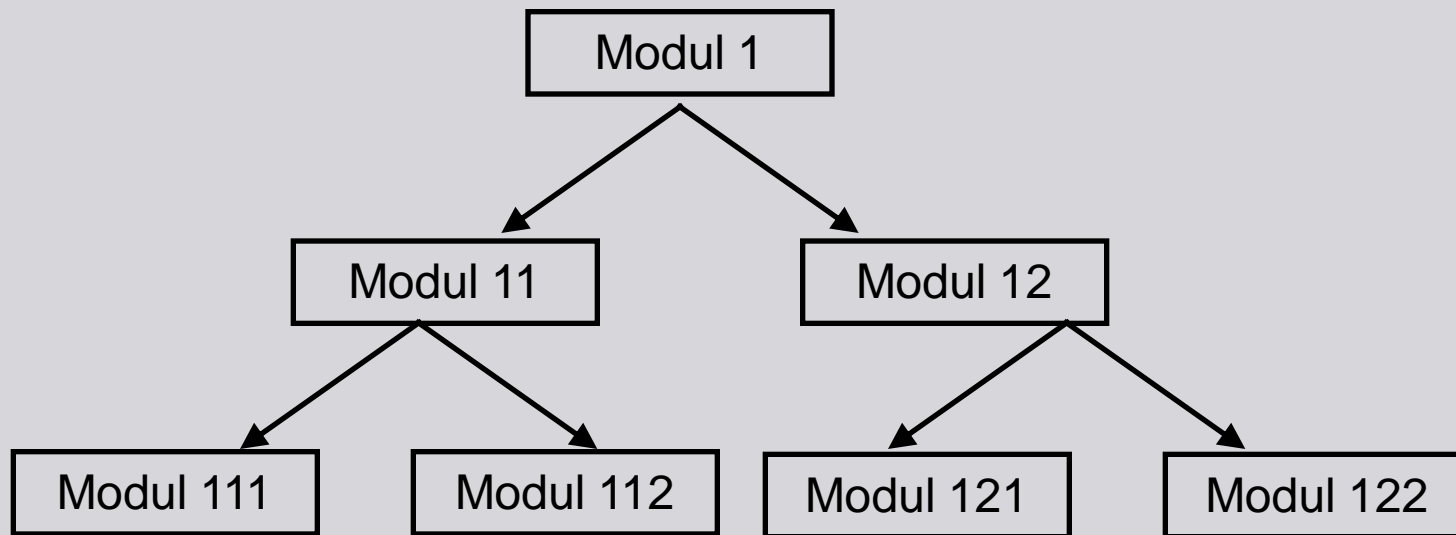
- Parameterwerte bei Aufrufen
- Werte globaler Variablen
- Aufrufreihenfolgen
- Zeitverhalten

Ohne dokumentierte SW Architektur ist ein sinnvoller Integrationstest nicht möglich!

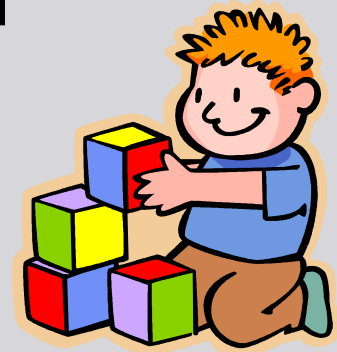


Test Phasen – Integrationsteststrategien

Beispiel

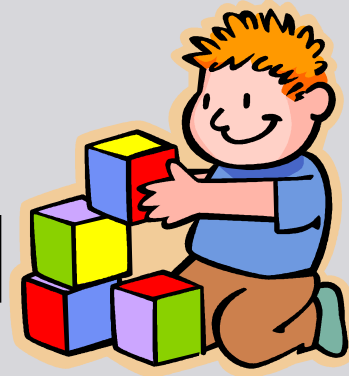
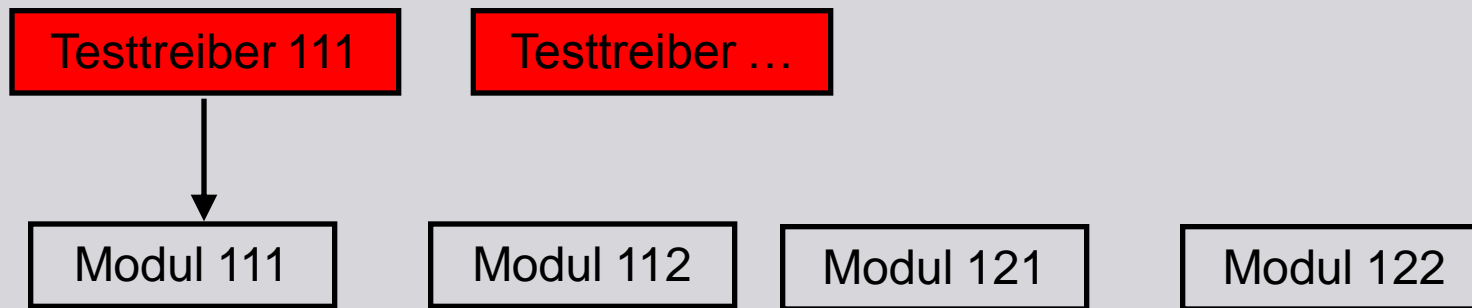


→ Aufrufbeziehung



Test Phasen – Bottom Up Integration (Theorie)

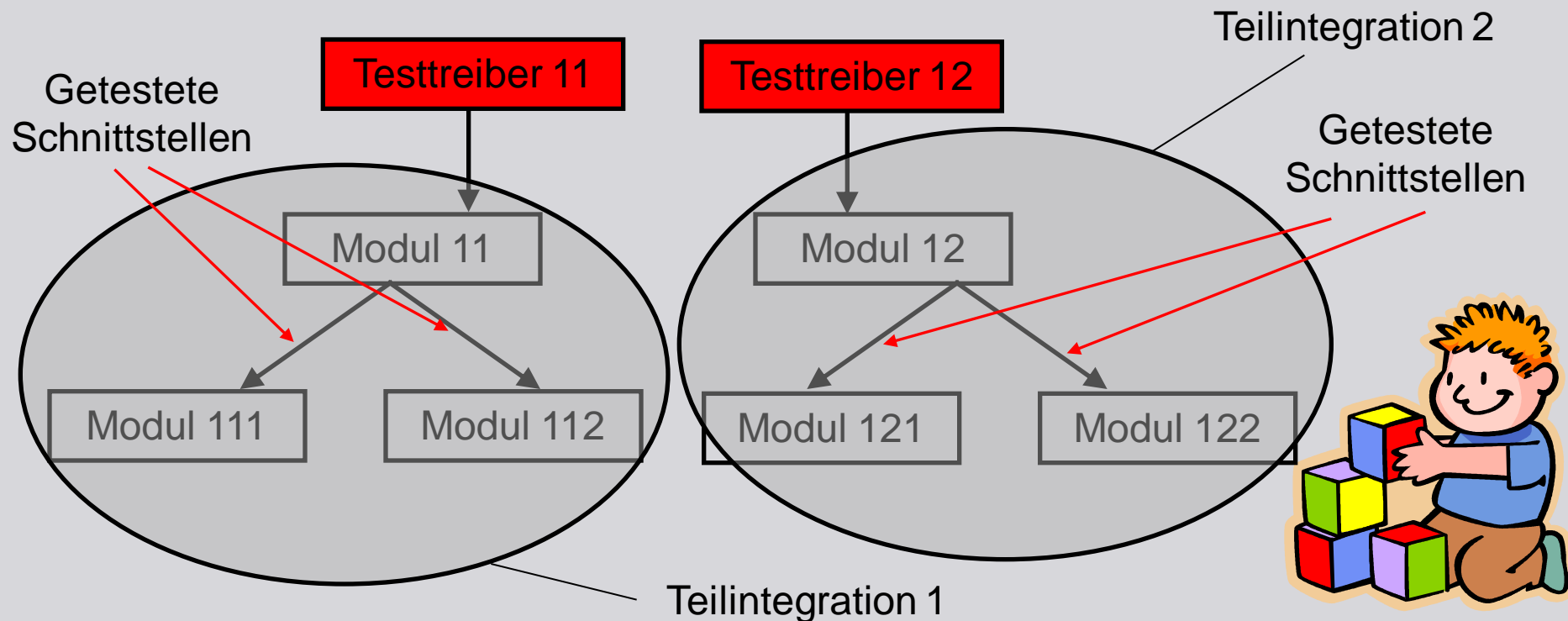
1. Schritt:
(Modul-)Test derjenigen Module, die keine weiteren aufrufen.
Benötigt werden Testtreiber, die den Aufrufer und ggf. die Umgebung simulieren



Test Phasen – Bottom Up Integration (Theorie)

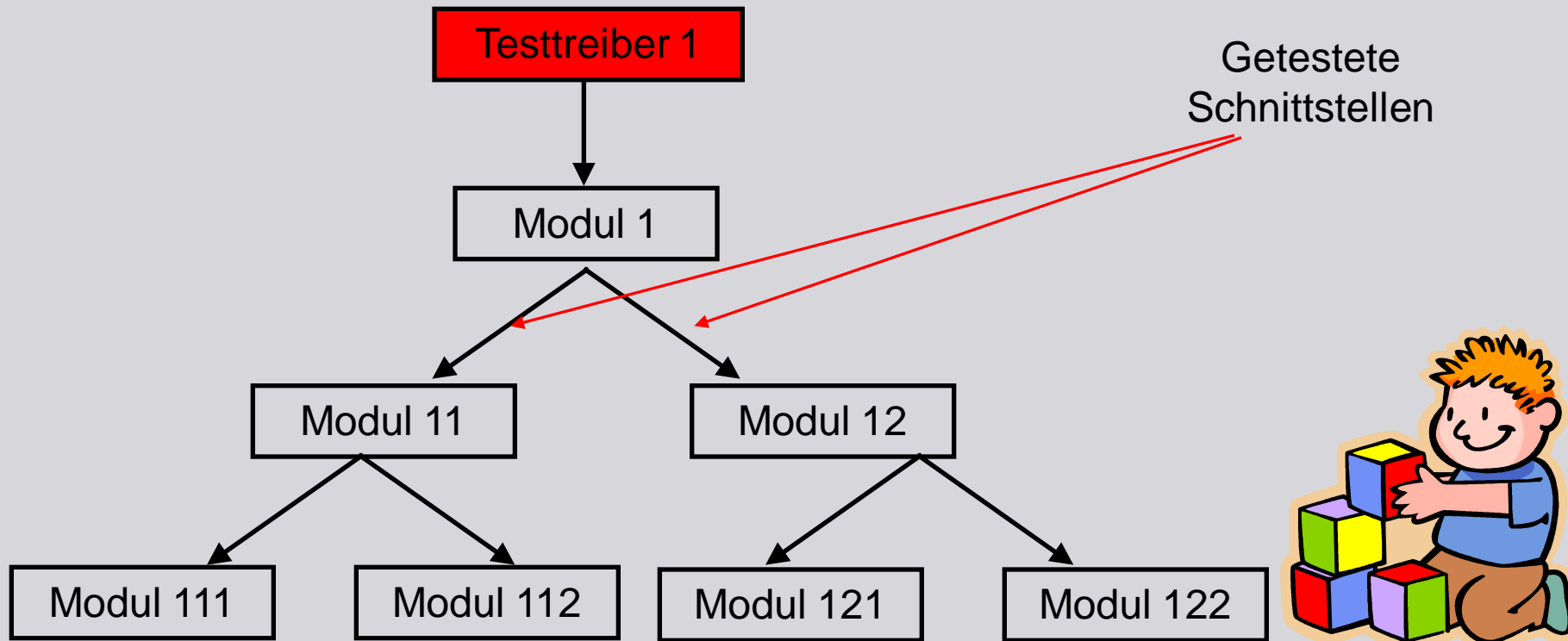
2. Schritt:

Integration der getesteten Module in die aufrufenden und Test der Integrationen. Benötigt werden wieder Testtreiber



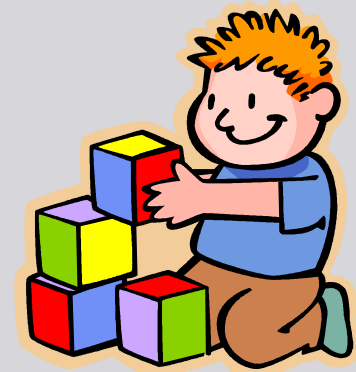
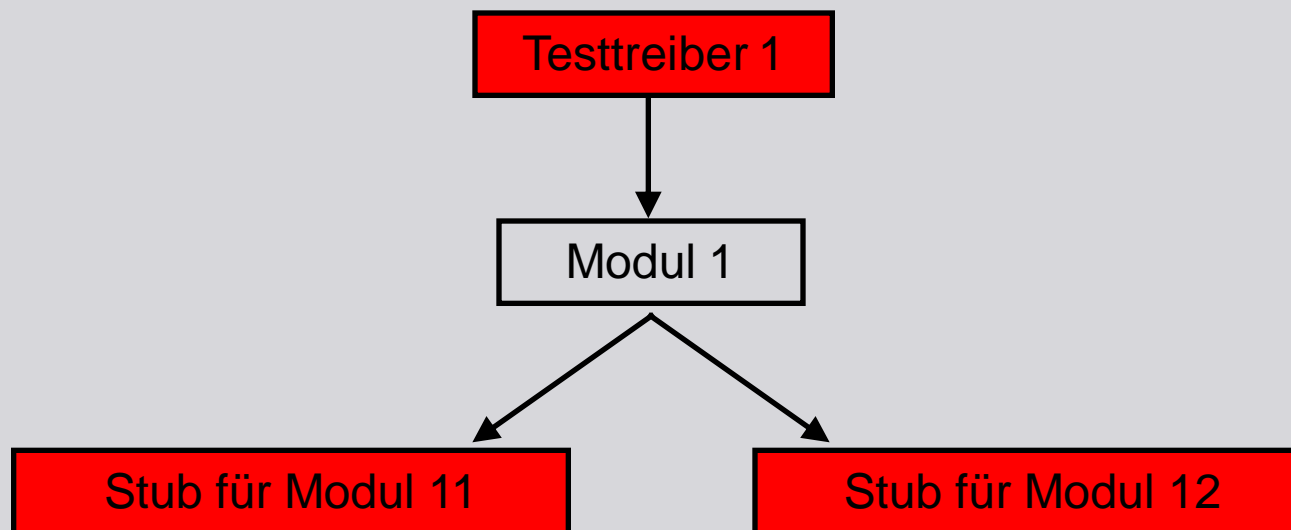
Test Phasen – Bottom Up Integration (Theorie)

3. Schritt:
Integration der Ergebnisse des letzten Integrationsschrittes zum Gesamtprogramm



Test Phasen – Top-Down Integration (Theorie)

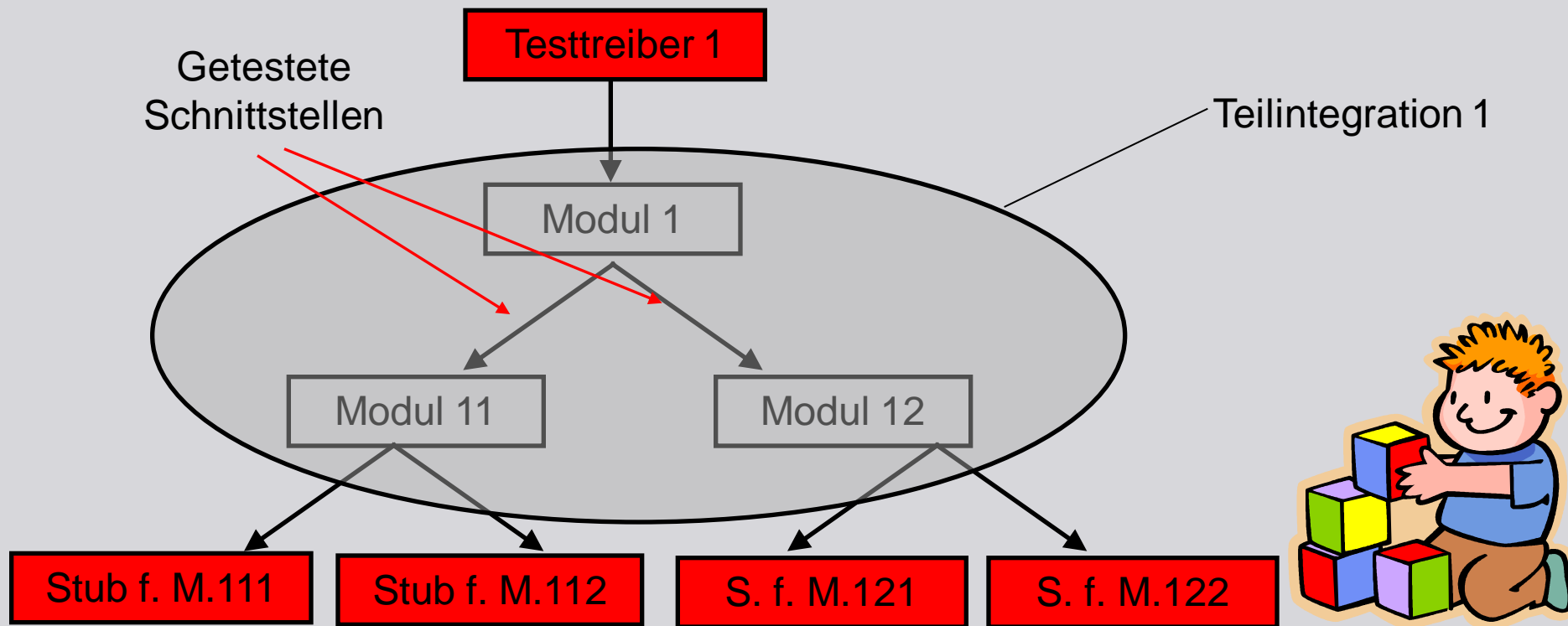
1. Schritt:
(Modul-)Test derjenigen Module, die von keinem anderen aufgerufen werden (main).
Benötigt werden Testtreiber, die den Aufrufer und Stubs, die die aufgerufenen Module simulieren



Test Phasen – Top-Down Integration (Theorie)

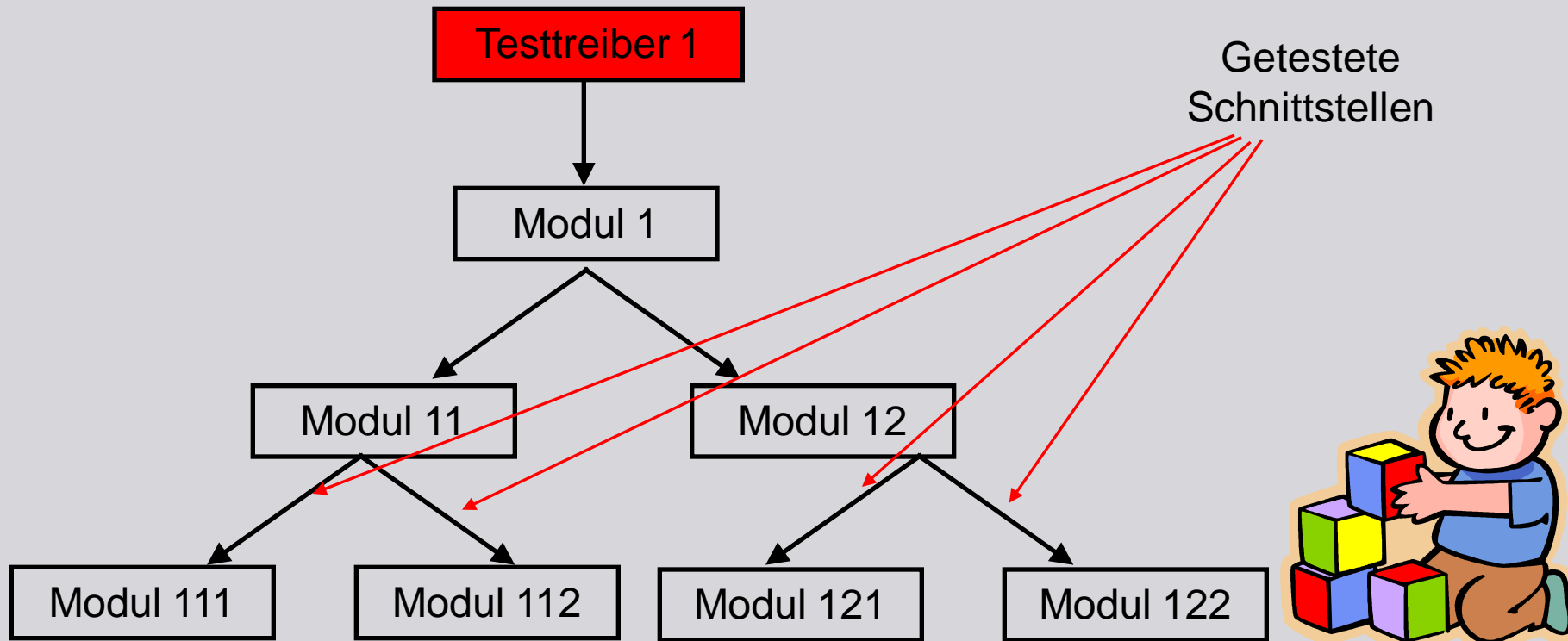
2. Schritt:

Integration der Module, die bisher durch Stubs ersetzt wurden. Module, die von den integrierten Modulen aufgerufen werden, werden wiederum simuliert.



Test Phasen – Top-Down Integration (Theorie)

3. Schritt:
Integration der zuletzt „gestubten“ Module zum
Gesamtprogramm



Test Phasen – Integrationstest - Anmerkungen

Top-Down Integration

Testvorgehen anpassen

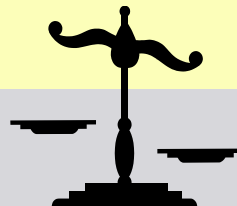
Bottom-Up Integration

- ☺ Augenmerk auf Kontrollfluss
- ☺ Grundgerüst frühzeitig testbar
- ☺ Ermöglicht Vorführung

- ☹ Benötigte **Stubs** können sehr aufwändig sein

- ☺ Augenmerk auf Komponenten
- ☺ Simulation "höherer Komponenten durch **Testtreiber**
- ☺ Flexibel im Ablauf

- ☹ Das Gesamtprogramm existiert erst, wenn das letzte Modul integriert ist



Test Phasen –Big Bang Integration

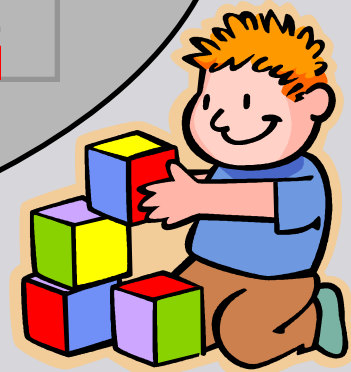
1 Integrationsschritt

Nachteile:

☹️ Mögliche Fehlerverdeckung

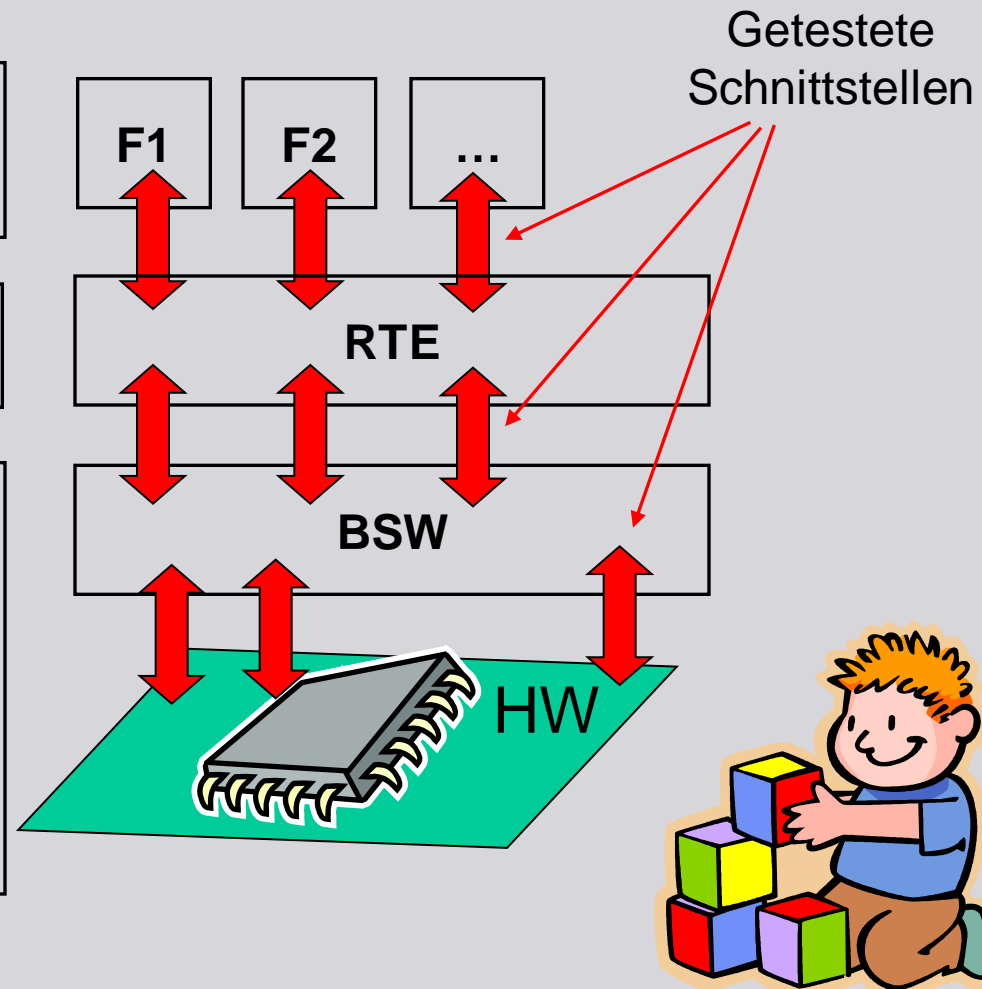
☹️ Fehler schwierig zu lokalisieren

☹️ Zahlreiche Abläufe und Schnittstellen zu beobachten



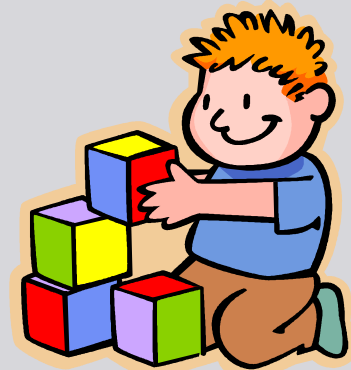
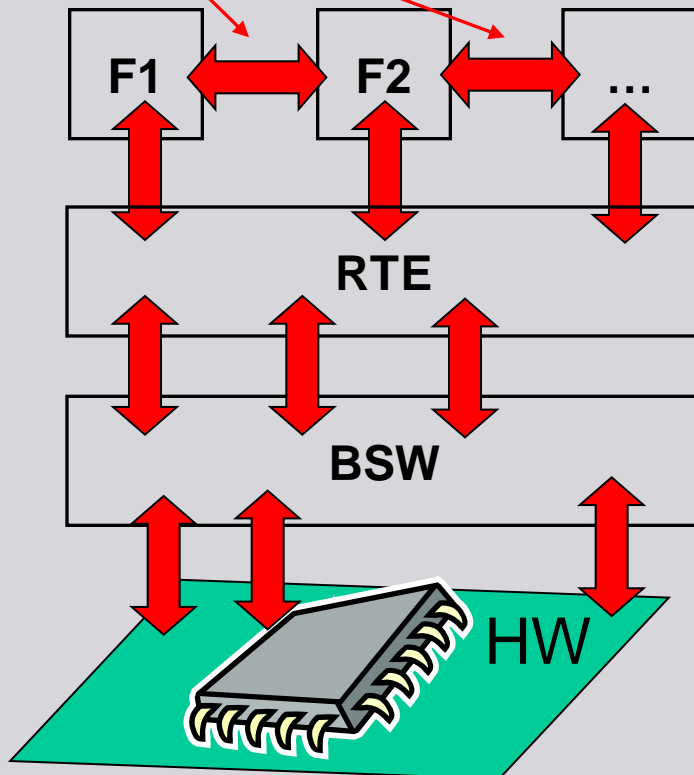
Test Phasen – Schichtweise Integration (Praxis)

3. Schritt:
Integration der Applikations-SW
Funktionen
2. Schritt:
Integration Runtime Environment
1. Schritt:
Integration des Betriebssystems (HW
nahen SW):
 - Scheduler, Memory Mgmt., I/O
Handling, Kommunikation: CAN,
LIN, Treiber
 - Test der HW-SW Interface



Test Phasen – Schichtweise Integration (Praxis)

Schrittweise Integration der Applikationsfunktionen F1, F2, ... Test der Schnittstellen zwischen den Funktionen



Inhalte

5.4 Test

5.4.1 Statischer Test

5.4.2 Dynamischer Test

5.4.3 Testphasen

5.4.3.1 Modultest

5.4.3.2 Integrationstest

5.4.3.3 Validierung

5.4.4. Abgrenzung SW-Test <-> Systemtest

5.4.5. Testtechniken und Teststrategien

...

Test Phasen – Valdierung

SW Validation ist der Test der kompletten SW

- gegen die Anforderungen
- unabhängig von der Hardware (speziell im Embedded Bereich)
- auch wenn der Test auf der Zielhardware durchgeführt wird

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4 Test

5.4.1 Statischer Test

5.4.2 Dynamischer Test

5.4.3 Testphasen

5.4.3.1 Modultest

5.4.3.2 Integrationstest

5.4.3.3 Validierung

5.4.4 Abgrenzung SW-Test <-> Systemtest

5.4.5 Testtechniken und Teststrategien

...

Test Phasen – Abgrenzung Systemtest

Weitere Testphasen parallel oder nach den SW Tests (speziell im Embedded Bereich)

- HW-SW Integration
- Komponentenintegration (Test des Zusammenspiels zwischen HW, SW und Mechanik)
- Komponentvalidierung (Test gegen die Anforderungen der Komponente, oft ähnlich den SW Anforderungen)
- Systemintegration (Test des Zusammenspiels von Systemkomponenten)
- Systemtest (Test eines kompletten Systems gegen die Systemanforderungen)

In der Praxis ist eine genaue Trennung zwischen diesen verschiedenen Testphasen manchmal schwierig

Test Phasen – Abgrenzung Systemtest

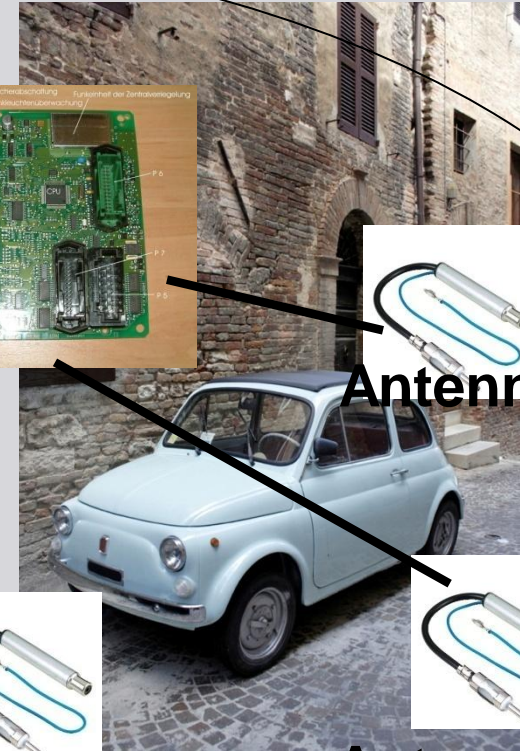
Beispiel System - Komponente



Funkschlüssel (Komponente)



Steuergerät (Komponente)



Antenne

Antenne

Antenne (Komponente)

System

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4 Test

...

5.4.4 Abgrenzung SW-Test <-> Systemtest

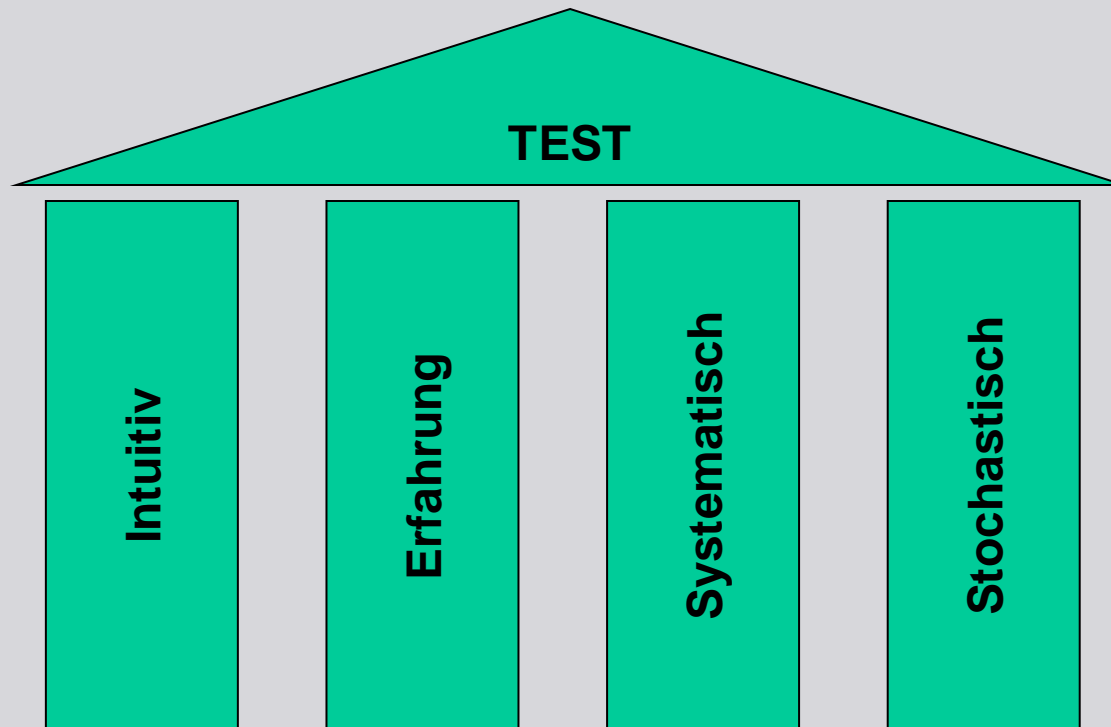
5.4.5 Testtechniken und Teststrategien

5.4.5.1. Black-Box

5.4.5.2. White-Box

...

Teststrategien



Intuitives Testen (Error Guessing)

Testfallspezifikation basiert auf einem intuitiven Ansatz:

- a) Anstatt eines systematischen Tests
z.B. Test eines Prototyps
- b) Zusätzlich zu systematischen Tests
- c) basierend auf der Erfahrung des Testers

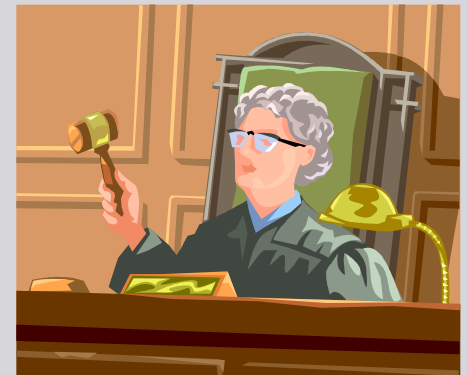


Ziel: Die “merkwürdigen” Fehler finden

Intuitives Testen (Error Guessing)

Wichtig!

**Undokumentiertes, “Ad Hoc Testen”
ohne spezifizierte Testfälle
und ohne Test Reports hat keine
rechtliche Bedeutung**



Erfahrung (Lessons learned)



• Checklisten

VALIDATION-Checklist			
Topics to be tested	Applicable	Non-applicable	Covered by
Functionality state after software initialization	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Chapter 4.5.5
Functionality at the first power ON (in case of a configuration in EEPROM)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Chapter 4.5.5
Timing constraint (response delay,...)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Chapter 4.5.1-4.5.3
Delay duration	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Chapter 4.5.1-4.5.3
accurate signal timing	<input type="checkbox"/>	<input checked="" type="checkbox"/>	-> SVVP CAN
Functionality must be tested with a cyclic test (durability test)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

• Fehlertracking

Testcases covering known SAR's	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
RVSGNBC~4 Laufzeitüberwachung (12.04.2002) : Implementiert aber noch nicht vollständig getestet			Covered by this Test
RVSGNBC~23 Laufzeit aktiviert -> Verdeck entriegeln -> (17.04.02) : Lampe und Summer geht nicht an			
RVSGNBC~47 Laufzeitüberwachung anpassen (15.04.02) : Zus. Faktor für Klappen und Sicherungen und Zähler bei Nichtansteuerung Decrementieren			

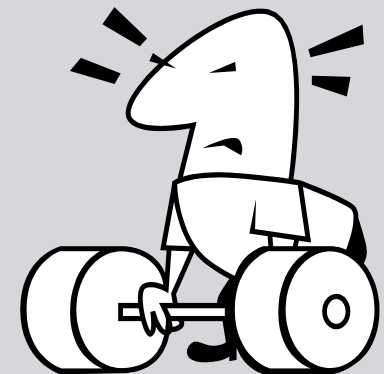
• Projektspezifische Themen z.B. Review Ergebnisse

Stochastische Tests / Stresstests

- Stress Test, insbesondere bei kritischen Laufzeiten
 - Test mit hoher Last (z.B. Interrupts, Kommunikation)
 - Test mit Überlast (→ Robustheit)
 - Gleichzeitiges Triggern der Inputs

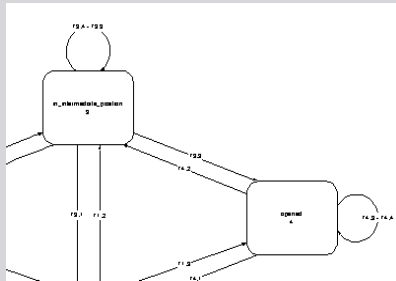
Beispiele (Automotive):

- KL 15 100 mal in 10s schalten
- Hohe Interrupt Last über CAN



Systematische Tests

- Ziel: Die “offensichtlichen” Fehler finden
 - z.B. Zustandsbasierter Test : Spezifikation



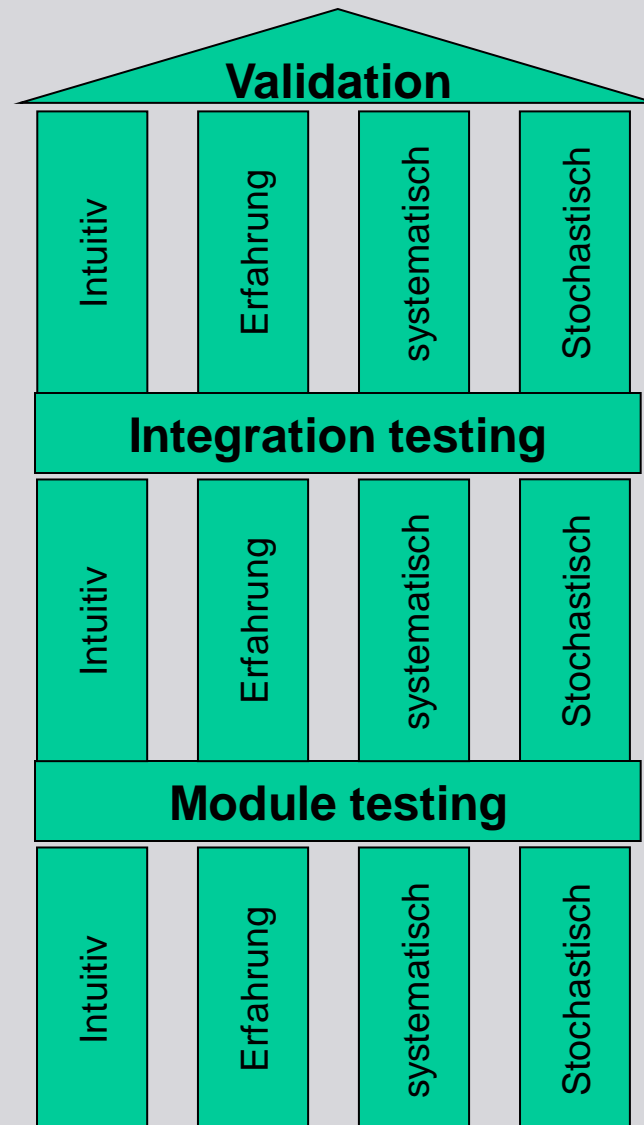
T3.1.1	$(ESL/R_Z == 1) \wedge (ESL/R_A == 1) \wedge (KI.15 == 1)$	Set Error Code "Inplausible switch position"
T3.1.2	$(ESL/R_Z == 1) \wedge (ESL/R_A == 1) \wedge (KI.15 == 0)$	-
T3.2	$RE\ ESL/R_Z == 1 \wedge (ESL/R_A == 0)$	Stop Timer "Timeout close latch"; Start Time "N_Time close latch"
T3.3	$RE\ ESL/R_A == 1 \wedge (ESL/R_Z == 0)$	Stop Timer "Timeout open latch"
T3.4	$FE\ TV_A == 1 \vee (TV_A == 1 \wedge TV_Z == 1) \vee (KI.15 == 0) \vee (Top\ NOT\ in\ state\ opened) \vee (V1 > 5\ km/h) \vee (V2 > 5\ km/h) \vee (Voltage\ NOK) \vee (Overload)$	Switch OFF output OPEN latch; Stop Timer "Timeout open latch"

- z.B. Zustandsbasierter Test : Testspezifikation



011	Clear error codes via K-Line. Switch of KI. 15. Do a HW Reset for power up, again.	(0)(0)(0)(0)(1)(0)(1)(0) 1 1 (0)(0)	Rising edge KI.15	T2.6	Opening and Closing: Error Code tbd "Inplausible switch position" is set	
012	Clear error codes via K-Line. KI. 15 already on	(0)(0)(0)(0)(1)(0)(1)(0) 1 1 (0)(0)	Do a HW Reset for power up, again.	T2.6	Opening and Closing: Error Code tbd "Inplausible switch position" is set	
013	Clear error codes via K-Line.	(0) 1 (0) 1 (1)(0)(1)(0)(0)(0)(0)	Do a HW Reset for power	T2.6	Closed and	

Das Testgebäude

**Alle Testlevels
sind Bausteine im
„Testgebäude“**



Test Techniken : Überblick

Black-Box	White-Box
 <ul style="list-style-type: none">▫ <i>Äquivalenzklassenanalyse</i>▫ <i>Grenzwertanalyse</i>▫ <i>Zustandsbasiertes Testen</i>▫ <i>Interface Testen</i>▫ Syntax Testen▫ ...	 <ul style="list-style-type: none">▫ <i>Statement Testen</i>▫ <i>Zweig/Decision Testen</i>▫ <i>Path Testen</i>▫ Datenflußtesten▫ Real-Time Testen▫ ...

Test Strategien

Black Box Test

(Synonym Funktionaler Test)

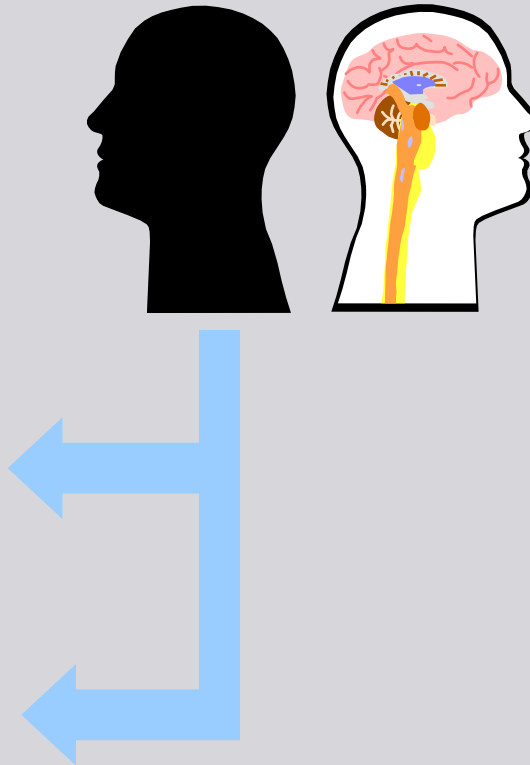
- Testfälle werden von der Spezifikation abgeleitet
- Design/ Implementierung werden nicht betrachtet

Positiver Test:

- Test SW unter den spezifizierten Bedingungen.

Negativer Test:

- Test der SW unter nicht spezifizierten Bedingungen (z.B. falscher Input, extreme Zeitbedingungen).
- Unterstützt Robustheitstests.



White Box Test

(Synonym
Strukturbasierter Test)

- Testfälle werden von der internen Struktur der SW abgeleitet (d.h. Design und Code)
- Keine Berücksichtigung der Spezifikation (nur in der Theorie!)

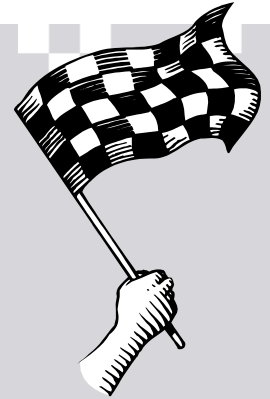
Testvollständigkeitskriterium

Eine Teststrategie **muss** ein formales **Test Vollständigkeitskriterium** beinhalten, das festlegt, wann der Test beendet ist.

Ein Testvollständigkeitskriterium muss messbar, formal definiert und dokumentiert sein (spätestens bei Testbeginn).

“Zu wenig Zeit”, “zu wenig Geld”, “wir benötigen die Software in zwei Stunden“ sind keine formalen Kriterien!

Der Erfüllungsgrad eines Testvollständigkeitskriteriums kann als Metrik für den Testfortschritt verwendet werden.



Testvollständigkeitskriterium

Eine Test **muss** ein formales **Testendekriterium** beinhalten, das festlegt, wann der Test beendet ist **und** die Software freigegeben werden kann:



- Testvollständigkeitskriterium erfüllt
- Keine kritischen Fehler
- Nur eine definierte Zahl nichtkritischer Fehler

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4 Test

...

5.4.4 Abgrenzung SW-Test <-> Systemtest

5.4.5 Testtechniken und Teststrategien

5.4.5.1 Black-Box

5.4.5.2 White-Box



Test Techniken - Black Box



Äquivalenzklassenanalyse (Equivalence Class Analysis)

Definition und Ziel:

Definiere Partitionen des Eingabebereichs und des Wertebereichs einer Funktion. Jede Partition enthält Werte, die von der Funktion auf die gleiche Art und Weise verarbeitet bzw. behandelt werden. Sowohl gültige als auch ungültige Wertebereiche werden so definiert. Diese Partitionen werden als Äquivalenzklassen bezeichnet

Bei Funktionen mit mehreren Inputparametern müssen geeignete Kombinationen ausgewählt werden.

Testdesign:

für jeden Testfall wird festgelegt:

- Die Eingabewerte,
- Die entsprechenden Partitionen,
- Das erwartete Testergebnis.

Testvollständigkeitskriterium:

Wenigstens ein Eingabe/Ausgabepaar aus jeder Äquivalenzklasse

Test Techniken - Black Box



Grenzwertanalyse (Boundary Value Analysis)

Definition und Ziel:

Verfeinerung der Äquivalenzklassenanalyse. Zusätzliche Betrachtung der Grenzen zwischen den Äquivalenzklassen

Testdesign:

Für jeden Testfall spezifiziere:

- Die Eingabewerte,
- Die entsprechenden Grenzen der Äquivalenzklassen,
- Das erwartete Testergebnis.

Testvollständigkeitskriterium:

Wenigstens ein Eingabe/Ausgabepaar aus jeder Äquivalenzklasse und Einausgabepaare an den jeweiligen Grenzbereichen der Äquivalenzklassen .

Beispiel: Äquivalenzklassen- und Grenzwertanalyse

- enum wochentag (Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag, Sonntag, Error)
- (C-) Function wochentag Datum_zu_Wochentag (int Tag, int Monat, int Jahr)
- Datum_zu_Wochentag berechnet den Wochentag zu einem gegebenen gültigen Datum (z.B. 12.3.01 ist ein Montag), falls das Datum korrekt ist und gibt Fehler zurück bei ungültigem Datum (z.B. 30.2.01)
- Weitere Bedingungen: Schaltjahr

(source: *Software Testing in the Real World*, Edward Kit)

Beispiel: Äquivalenzklassen- und Grenzwertanalyse

Lösung 1

- Äquivalenzklassen für Tag $[-\infty .. 0]$, $[1 .. 31]$, $[32 .. \infty]$
- Äquivalenzklassen für Monat $[-\infty .. 0]$, $[1 .. 12]$, $[13 .. \infty]$
- Äquivalenzklasse für Jahr $[-\infty .. \infty]$

➔ Grenzwerte für Tag: 0, 1, 31, 32

➔ Grenzwerte für Monat: 0, 1, 12, 13

➔ Grenzwerte für Jahr: keine

➔ Weitere Randbedingungen z.B.:

- 31 Tage im Januar, März, ...
- 30 Tage im April, Juni, ...
- 28/29 Tage im Februar

Beispiel: Äquivalenzklassen- und Grenzwertanalyse

Lösung 2

- Äquivalenzklassen für Tag: $[-\infty .. 0; 32 .. \infty]$, $[1 .. 28]$, $[29]$, $[30]$, $[31]$
- Äquivalenzklassen für Monat: $[-\infty .. 0]$, $[1 .. 12]$, $[13 .. \infty]$
- Äquivalenzklasse für Jahr: $[-\infty .. -1]$, $[0]$, $[1 .. \infty]$

➔ Grenzwerte für Tag : 0, 1, 31, 32

➔ Grenzwerte für Monat : 0, 1, 12, 13

➔ Grenzwerte für Jahr : -1, 0, 1

➔ Weitere Randbedingungen z.B.:

- 31 Tage im Januar, März, ...
- 30 Tage im April, Juni, ...
- 28/29 Tage im Februar

Übung



•Äquivalenzklassen- und Grenzwertanalyse?

Die Geldbußen für Geschwindigkeitsübertretungen im Straßenverkehr werden im Rahmen einer 2012 durchgeführten Reform folgendermaßen festgesetzt:

- Geschwindigkeitsübertretung bis einschließlich 10km/h 20€ Bußgeld
- Geschwindigkeitsübertretung über 10km/h bis einschließlich 20km/h 60€ Bußgeld
- Geschwindigkeitsübertretung über 20km/h bis einschließlich 40km/h 100€ Bußgeld
- Geschwindigkeitsübertretung über 40km/h bis einschließlich 60km/h 300€ Bußgeld
- Geschwindigkeitsübertretung über 60km/h 1000€ Bußgeld
- Autofahrern mit Jahresbruttogehalt bis einschließlich 20000€ zahlen einfaches Bußgeld
- Autofahrern mit Jahresbruttogehalt zwischen 20000€ bis einschließlich 40000€ zahlen doppeltes Bußgeld
- Autofahrern mit Jahresbruttogehalt über 40000€ zahlen dreifaches Bußgeld
- Bei Geschwindigkeitsübertretungen unter Alkoholeinfluss wird das zusätzlich Bußgeld mit dem Faktor $(1 + \text{Promillegehalt im Blut})$ multipliziert.

Führen Sie eine Äquivalenzklassen- und Grenzwertanalyse durch und definieren Sie eine Menge von Testfällen, die alle Klassen und ihre Grenzen mit mindestens einem Testfall abdeckt.

15 min, arbeiten Sie ggf. zusammen mit einem Partner

Test Techniken - Black Box



Zustandsbasierter Test (State Transition Testing)

Definition und Ziel:

Basis des Tests ist ein Zustandsmodell, das das Verhalten der Software spezifiziert und das in der Software umgesetzt ist. Getestet werden die Zustandsübergänge (Transitionen), ausgelöst durch die Ereignisse, die die Übergänge auslösen und die Aktionen, die auf Grund der Transitionen ausgeführt werden.

Test Design:

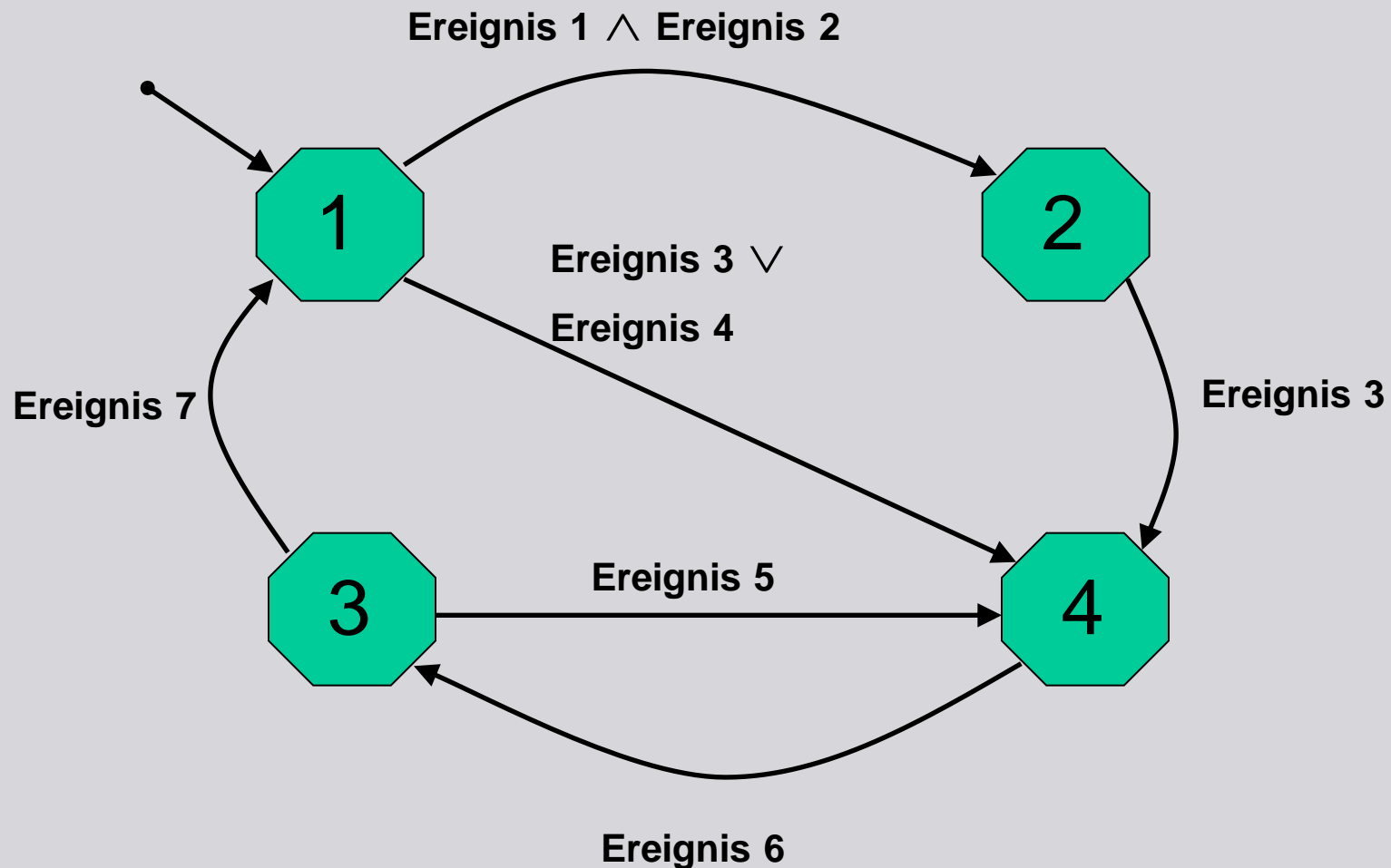
Für jeden Testfall wird spezifiziert:

- der Ausgangszustand der zu testenden Software
- das Ereignis, das einen Zustandsübergang auslöst
- die erwartete Reaktion (z.B. Ausführen einer Aktion)
- der erwartete Folgezustand

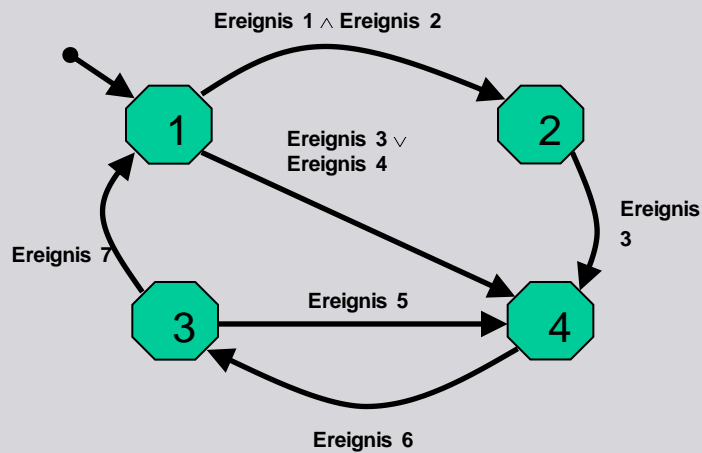
Testvollständigkeitskriterium:

100% Überdeckung des Zustandsautomat (d.h. Ausführen aller möglichen Transitionen)

Beispiel: Zustandsbasierter Test



Beispiel: Zustandsbasierter Test



Zustandsüberdeckung durch z.B.

- Start der Automaten
- Ereignis 1 und 2
- Ereignis 3
- Ereignis 6
- Ereignis 7
- Ereignis 3
- Ereignis 6
- Ereignis 5

Test Techniken - Black Box



Schnittstellen Test (Interface Testing)

Definition und Ziel:

Der Schnittstellen Test basiert auf SW Architektur bzw. Feindesign, die die Schnittstellen zwischen SW Komponenten beschreiben (z.B. Definition, Ablauf, Timing). Der Test soll sicherstellen, dass die Schnittstellen wie spezifiziert umgesetzt wurden.

Test Design:

- Auswahl der zu testenden SW Komponenten
- Auswahl der zu testende(n) Schnittstelle(n) zwischen diesen Komponenten
- Datenbereiche und Typen der Schnittstelle(n)
- Art und Anzahl der Parameter an der Schnittstelle
- Die Methode der Datenübergabe.
- Die Abläufe für den Schnittstellentest

Test Techniken - Black Box



Schnittstellen Test (Interface Test)

Testvollständigkeitskriterium:

- Theoretisch alle Schnittstellen (praktisch nicht durchführbar)
- in der Praxis Auswahl
 - Wichtiger
 - kritischer
 - Komplexer
 - Schnittstellen (ggf. intuitive Auswahl)

Nur für den Integrationstest

Test Techniken - Black Box



Entscheidungstabellentest

Definition und Ziel:

Methode zur Auswahl von Testfällen bei Systemen mit vielen Eingabeparametern. Eingabeparameter werden geeignet verknüpft um eine bestimmte Aktionen hervorzurufen. (Ursache -> Wirkung)

In einer Tabelle werden alle Eingabeparameter und ihre Werte mit allen Aktionen zu Testfällen verknüpft.

Ziel: relevante, wichtige Kombinationen von Eingaben testen. „relevant“ oder „wichtig“ im Sinn von möglichen Fehlerwirkungen.

Test Techniken - Black Box

Beispiel: Entscheidungstabellentest

- Anlassen eines Fahrzeugs mit Automatikgetriebe, Startknopf und Schlüssel gesteckt:

Um das Fahrzeug anzulassen sind mehrere Aktionen nötig:

- 1.) Schlüssel ist gültig
- 2.) Wählhebel in Parkposition
- 3.) Bremspedal ist getreten
- 4.) Startknopf wird gedrückt

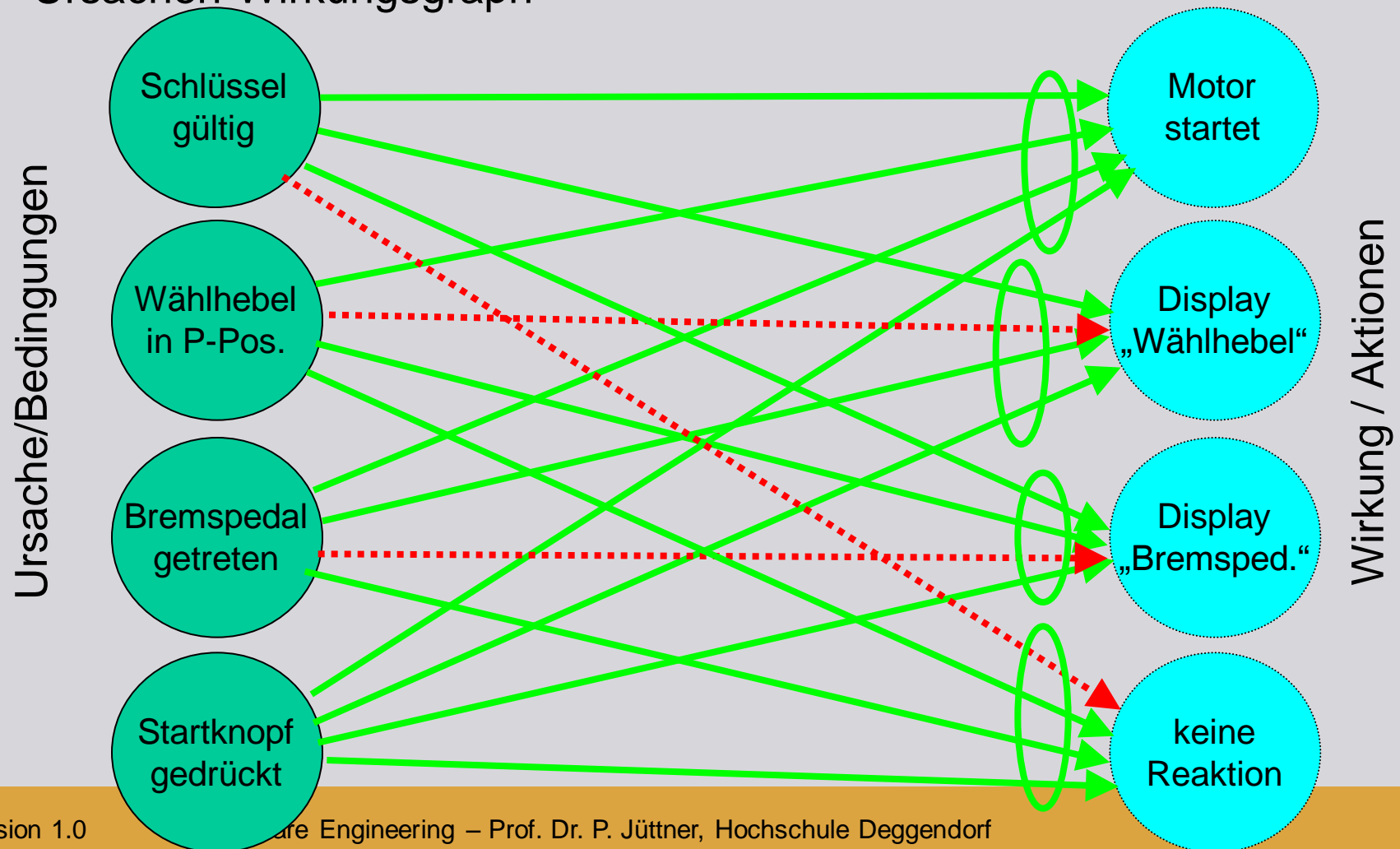
Folgende Aktionen des Fahrzeugs sind möglich:

- 1.) Motor startet
- 2.) Motor startet nicht, Displayausgabe „Bremspedal treten“
- 3.) Motor startet nicht, keine Displayausgabe
- 4.) Motor startet nicht, Displayausgabe „Wählhebel in Parkposition bringen“

Test Techniken - Black Box

Beispiel: Entscheidungstabellentest

- Ursachen-Wirkungsgraph



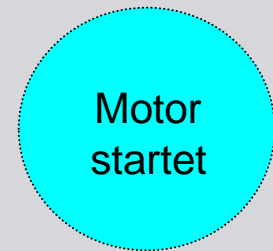
Test Techniken - Black Box

Beispiel: Entscheidungstabellentest

- Ursachen-Wirkungsgraph Legende



Ursache/Bedingungen



Wirkung / Aktionen

Ursache liegt nicht vor /
Bed. nicht erfüllt

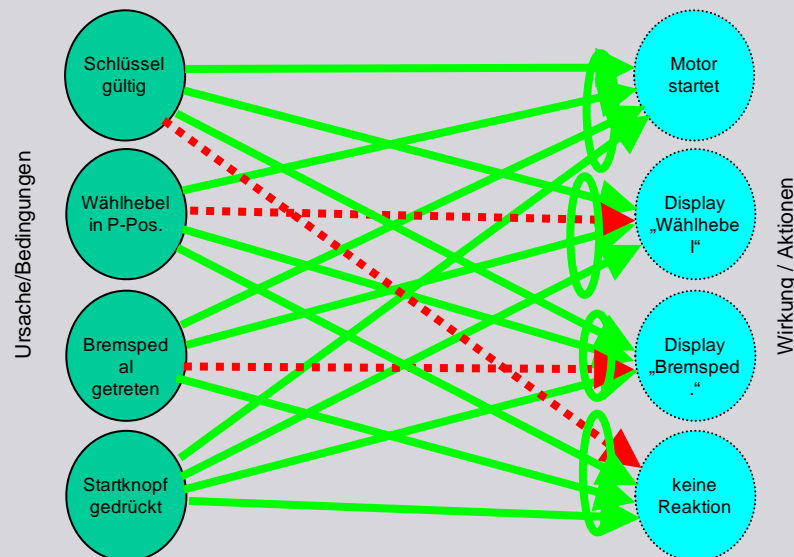
Ursache liegt vor /
Bedingung erfüllt

logische Und-Verknüpfung
von Ursachen/Bedingungen

Test Techniken - Black Box

Beispiel: Entscheidungstabellentest

- Ursachen-Wirkungsgraph
 - verdeutlicht, welche Bedingungen erfüllt sein müssen, um eine bestimmte Aktion oder Wirkung hervorzurufen
 - wird umgeformt in eine Entscheidungstabelle, aus der wiederum die Testfälle abgeleitet werden.



6		Testfall 1	Testfall 2	Testfall 3	Testfall 4
Bedingungen	Schlüssel gültig	nein	ja	ja	ja
	Wählhebel in P-Position	-	nein	ja	ja
	Bremspedal getreten	-	-	nein	ja
	Startknopf gedrückt	-	ja	ja	ja
Aktionen					
	Motor startet	nein	nein	nein	ja
	Display „Wählhebel“	nein	ja	nein	nein
	Display „Bremsped.“	nein	nein	ja	nein
	keine Reaktion	ja	nein	nein	nein

Test Techniken - Black Box

Beispiel: Entscheidungstabellentest

- Regeln für Entscheidungstabelle (nach Liggesmeyer)
 - 1.) Wähle eine Wirkung / Aktion aus
 - 2.) Finde Kombinationen im Graph, die diese Wirkung / Aktion auslösen bzw. nicht auslösen
 - 3.) Erstelle jeweils eine Spalte für alle gefundenen Ursachenkombinationen
 - 4.) Überprüfen auf und ggf. Entfernen von Redundanzen, unmöglichen oder unnötigen Kombinationen

Test Techniken - Black Box

Beispiel: Entscheidungstabellentest

Entscheidungs- tabelle		Testfall 1	Testfall 2	Testfall 3	Testfall 4
Bedingungen	Schlüssel gültig	nein	ja	ja	ja
	Wählhebel in P-Position	-	nein	ja	ja
	Bremspedal getreten	-	-	nein	ja
	Startknopf gedrückt	-	ja	ja	ja
Aktionen					
	Motor startet	nein	nein	nein	ja
	Display „Wählhebel“	nein	ja	nein	nein
	Display „Bremspedal“	nein	nein	ja	nein
	keine Reaktion	ja	nein	nein	nein

Test Techniken - Black Box



Entscheidungstabellentest

Testvollständigkeitskriterium:

- **Von jeder Spalte der Entscheidungstabelle wird ein Testfall ausgeführt**

Vor-/Nachteile

- + systematisches, formales Vorgehensweise**
- u.U. viele Testfälle -> unübersichtlicher Graph bzw. Tabelle**

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4 Test

...

5.4.4 Abgrenzung SW-Test <-> Systemtest

5.4.5 Testtechniken und Teststrategien

5.4.5.1. Black-Box

5.4.5.2. White-Box

...



Test Techniken - White Box



Statement Test

Definition und Ziel:

Der Statement Test basiert auf der Struktur der Software. Ziel ist es, alle Statements (Anweisungen) der zu testenden SW beim Test wenigstens einmal auszuführen.

Test Design:

Für jeden Testfall wird spezifiziert:

- Die Inputs der zu testenden SW Komponente
- Die Statements, die auszuführen sind
- Der erwartete Output

Testvollständigkeitskriterium:

Erreichen einer Testabdeckung von 100% aller Statements (C_0 -Abdeckung)

Überwiegend für Modultests, ohne entsprechende Tools nicht sinnvoll durchführbar

Test Techniken - White Box



Zweig/Bedingungs Test (Branch/Decision Test)

Definition und Ziel:

Der Zweig/Bedingungstest basiert auf der Struktur der Software. Ziel ist es, bestimmte Zweige der zu testenden SW (d.h. Ergebnisse der Auswertung einer Bedingung) auszuführen.

(eine Bedingung ist ein ausführbares Statement, das den Programmablauf abhängig von einer logischen Entscheidung bei einem anderen Statement fortsetzt.)

Test Design:

Für jeden Testfall wird spezifiziert:

- Die Inputs der zu testenden SW Komponente
- Die Programmzweige, die auszuführen sind
- Der erwartete Output

Testvollständigkeitskriterium:

Erreichen einer Testabdeckung von 100% aller Zweige (C_1 -Abdeckung)

Überwiegend für Modultests, ohne entsprechende Tools nicht sinnvoll durchführbar

Test Techniken - White Box



Pfad Test (Path Test)

Definition und Ziel:

Der Pfad Test basiert auf der Struktur der Software. Ziel ist es, bestimmte Pfade der zu testenden SW auszuführen.

(ein Pfad ist eine nacheinander ausgeführte Folge von Statements von Start der SW bis zur Beendigung)

Test Design:

Für jeden Testfall wird spezifiziert:

- Die Inputs der zu testenden SW Komponente
- Die Programmpfade, die auszuführen sind
- Der erwartete Output

Testvollständigkeitskriterium:

Erreichen einer Testabdeckung von 100% aller Pfade (C_2 -Abdeckung)
(In der Praxis ist eine C_2 -Abdeckung, fast nie erreichbar).

Test Techniken - White Box



Weitere White Box Test Techniken

Einfache Bedingungsüberdeckung

Jede **atomare Teilbedingung** hat beim Test einmal den Wert wahr und einmal den Wert falsch angenommen.

Mehrfachbedingungsüberdeckung

Jede Kombination der atomaren Teilbedingungen wird beim Test berücksichtigt

→ Aufwändig, da u.U. viele Testfälle (2^N bei N atomaren Bedingungen)

Minimale Mehrfachbedingungsüberdeckung

Wie Mehrfachbedingungsüberdeckung, jedoch nur Kombinationen auswählen, die den Wahrheitswert der Gesamtbedingung beeinflussen.

Übung

- Was bringt eine C_0 bzw. C_1 Überdeckung?
- Wo unterscheiden sich C_0 - (Statement-) und C_1 (Zweig-) Überdeckung?

Warum sind $C_{0/1}$ Überdeckungen sinnvoll, was bringen sie?
Finden Sie ein Codebeispiel, bei dem eine C_0 Überdeckung keine C_1 Überdeckung impliziert?

10 min, arbeiten Sie ggf. zusammen mit einem Partner



Test Techniken



Anmerkungen (1)



- Es gibt keine *beste Test Technik* (weder wissenschaftlicher Beweis noch industrieller Konsens)
- Alle Techniken haben Vor- und Nachteile
- Manche Techniken sind nur für eine bestimmte Testphase und oder Anwendungsgebiet sinnvoll
- Zusätzliche testrelevante SW ist oftmals erforderlich (z.B. Testrahmen)
- Tools sind notwendig
- “Nicht testbare” Software, z.B. sehr HW-nahe SW

Eine sinnvolle Kombination von Testtechniken ist erforderlich

⇒ Tool Evaluierung, Training, Einarbeitung sind sehr wichtig

Test Techniken



Anmerkungen (2)



Kombination von Test Techniken beim Modultest

- Ausführung eines Black-Box Tests, Messung der White-Box Überdeckung
- Falls White-Box Testendekriterium erreicht, Test beenden
- Andernfalls weitere White-Box Testfälle definieren und ausführen, ggf. auch Test abbrechen und analysieren, warum geplante Überdeckung noch nicht erreicht (toter Code!)

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4.6 Weitere Begriffe – Regressions- und Abnahmetest

5.4.7 Test Dokumentation

5.4.8 Testen im Automotive Umfeld

5.4.9 Reduktion des Testumfangs

5.4.10 Exkurs: Testen objektorientierter Software

5.4.11 Exkurs: Testdatengenerierung aus UML Modellen

5.4.12 Exkurs: Modellbasiertes Testen

5.4.13 Exkurs: Testen grafischer Bedienoberflächen

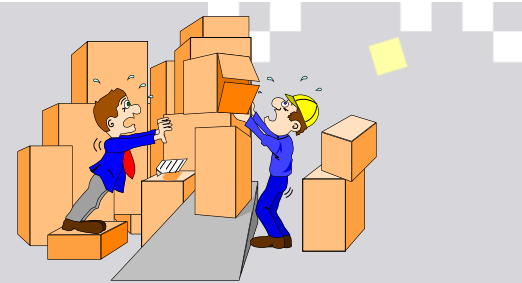
5.4.14 Do's und Dont's, Erfahrungen, Probleme

Test Prozess Weitere Begriffe

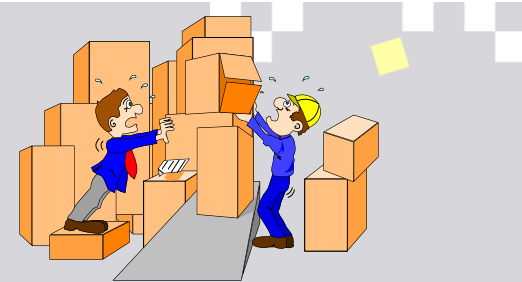
Regression Test

Wiederholung eines Tests (normalerweise nach einer Änderung), der zeigen soll, dass sich das Verhalten der Software nicht unbeabsichtigt geändert hat. [Beizer].

D.h. Unveränderte Teile der SW sollen sich so verhalten, wie zuvor, Änderungen wie spezifiziert.



Test Prozess Weitere Begriffe



Akzeptanztest (Abnahmetest)

Formal Test der es ermöglicht, dass ein Benutzer, ein Kunde oder eine dazu autorisierte Instanz eine SW abnimmt d.h. zur Verwendung freigibt [s. auch IEEE Std 610.12-1990].

Dieser Test wird normalerweise am Ende der Entwicklung durchgeführt, ggf. zusammen mit dem Kunden basierend auf einer eigenen Testspezifikation.

Im Gegensatz zu anderen Tests ist es hier nicht das Ziel Fehler zu finden.



Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

5.4.6 Weitere Begriffe – Regressions- und Abnahmetest

5.4.7 Test Dokumentation

5.4.8 Exkurs: Testen im Automotive Umfeld

5.4.9 Reduktion des Testumfangs

5.4.10 Exkurs: Testen objektorientierter Software

5.4.11 Exkurs: Testdatengenerierung aus UML Modellen

5.4.12 Exkurs: Modellbasiertes Testen

5.4.13 Exkurs: Testen grafischer Bedienoberflächen

5.4.14 Do's und Dont's, Erfahrungen, Probleme

Test Dokumentation - Dokumente

*Warum soll ich
meine Tests
dokumentieren
?*



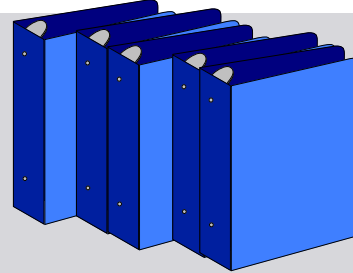
Ziele

- ✓ Hinterfragen des eigenen Tuns
- ✓ Kommunikation mit anderen
- ✓ Festhalten von Arbeitsergebnissen

Nutzen

- Sinnvoller Einsatz von Testmethoden und -strategien
- Nachweis gegenüber dem Kunden und / oder Gesetz
- Wieder verwendbare Tests (Regressionstest)
- Wiederverwendung für andere Projekte
- Verbesserte Testplanung und Testausführung
- Weiterverwendung von Testergebnissen (Metriken)

Test Dokumentation - Dokumente

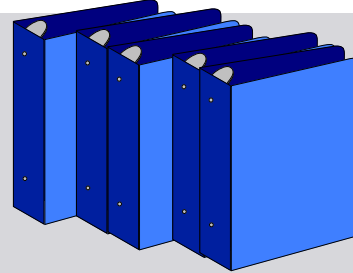


Teststrategie:

Beschreibt das generellen Testvorgehen im Projekt z.B.

- Verantwortlichkeiten
- Testphasen
- Verwendete Teststrategien und –methoden
- Testvollständigkeitskriterien
- Verwendete Tools
- Tailoring

Test Dokumentation - Dokumente



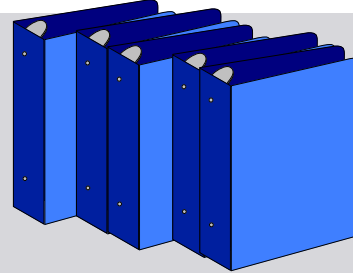
Testspezifikationen:

Testphasenspezifische Dokumentation der Testfälle, in der sich jeder Testfall eindeutig identifizieren läßt z.B.

- Modultestspezifikation
- Integrationstestspezifikation
- Traceability zu Anforderungen (ggf. in Anforderungen dokumentieren)

Testspezifikationen können (ganz oder teilweise) durch geeignete Testskripts für Testtools ersetzt werden.

Test Dokumentation - Dokumente



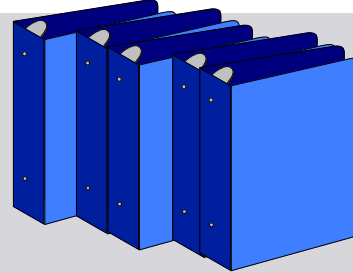
Test Log:

Detailliertes Ergebnis aller ausgeführten Testfälle einer Testdurchführung. Das Ergebnis jedes einzelnen Testfalls ist dokumentiert, z.B.

- **Testfallname/-nummer**
- **Ergebnis des Testfalls (gut/schlecht)**
- **ggf. Testabdeckung des Testfalls bzgl. Endekriterium**
- **ggf. Dauer / Ressourcenverbrauch**



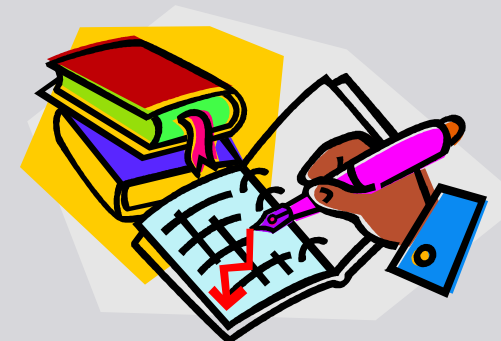
Test Dokumentation - Dokumente



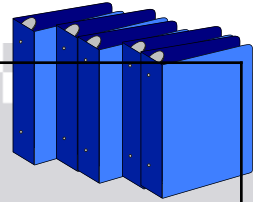
Test Report:

Zusammenfassung der Testlogs einer Testdurchführung, z.B:

- Anzahl ausgeführter Testfälle
- Anzahl durchgelaufener Testfälle
- Anzahl nicht durchgelaufener Testfälle
- ggf. Abdeckung Testendekriterium
- ggf. Dauer / Ressourcenverbrauch



Test Dokumentation - Dokumente



Beispiel Test Report:

Test Report SW Validierung	
Projekt	Karosseriesteuergerät XYZ
getestete Software	Karosseriesteuergerät XYZ (Komplettsoftware)
Version	3.7
Datum des Tests	3.8.2008
Dauer des Tests	50 min (automatischer Durchlauf)
Tester	Hans Meier
Anzahl Testfälle gesamt	350
Anzahl Testfälle durchgeführt	350
Anzahl Testfälle OK	325
Anzahl Testfälle nicht OK	25
Testende Kriterium erreicht	Nein (Testfälle nicht OK)

Zum Schluss dieses Abschnitts ...

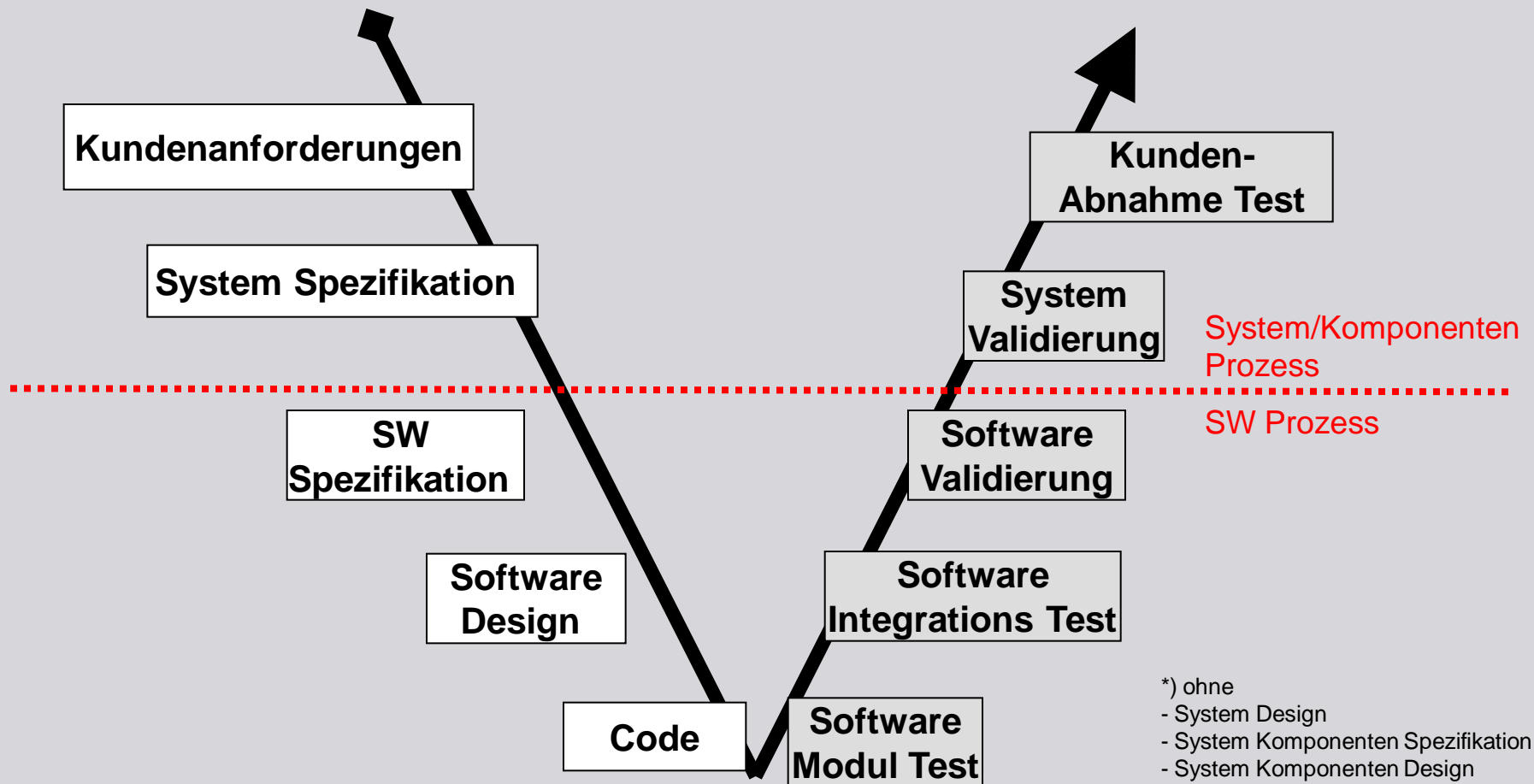
Noch Fragen ??

Inhalte

- 5.4.6 Weitere Begriffe – Regressions- und Abnahmetest
- 5.4.7 Test Dokumentation
- 5.4.8 Exkurs: Testen im Automotive Umfeld**
- 5.4.9 Reduktion des Testumfangs
- 5.4.10 Exkurs: Testen objektorientierter Software
- 5.4.11 Exkurs: Testdatengenerierung aus UML Modellen
- 5.4.12 Exkurs: Modellbasiertes Testen
- 5.4.13 Exkurs: Testen grafischer Bedienoberflächen
- 5.4.14 Do's und Dont's, Erfahrungen, Probleme

Exkurs: Automotive Projekte und Software

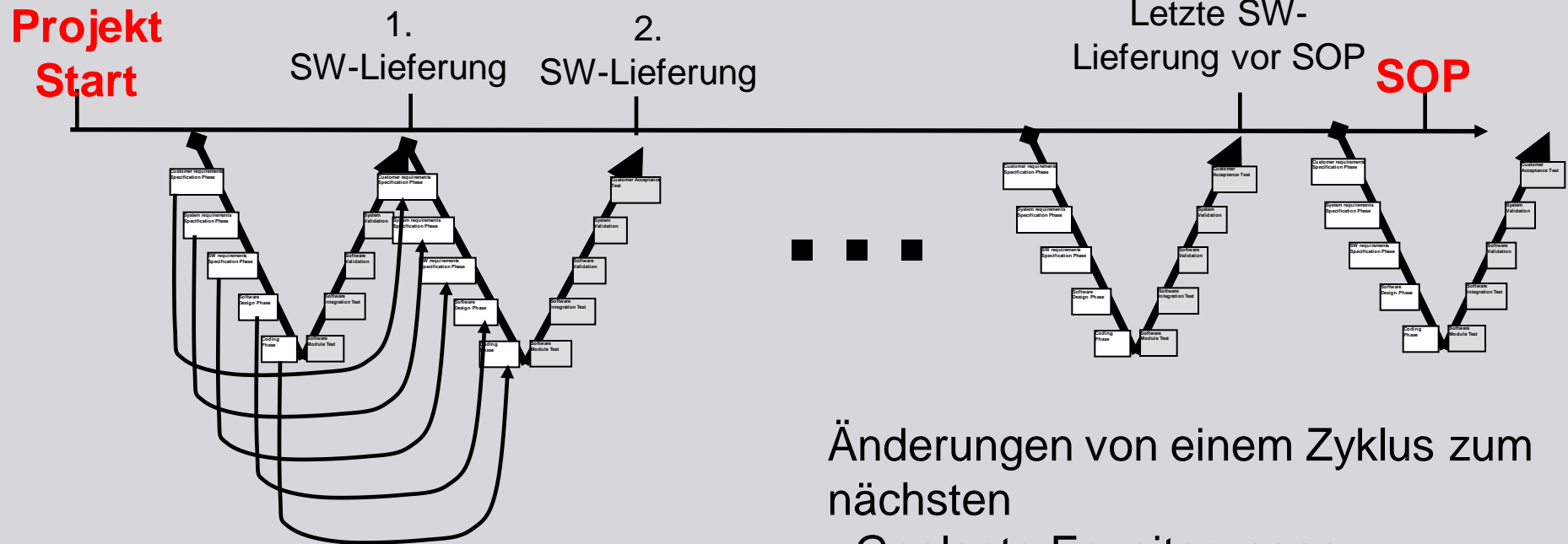
Integrierte System / SW Entwicklungsprozesses (vereinfacht*)



*) ohne

- System Design
- System Komponenten Spezifikation
- System Komponenten Design
- System Komponenten Test
- Bug Fix Phasen nach Testphasen

Exkurs: Automotive Projekte und Software



Ergebnisse eines Entwicklungszyklus werden im nächsten weiterverwendet

Änderungen von einem Zyklus zum nächsten

- Geplante Erweiterungen
- Änderungswünsche
- Fehlerbehebung

Exkurs: Automotive Projekte und Software

Folgerungen:

- nicht jede Softwarelieferung kann und/oder muss komplett getestet werden (s. Reduktion des Testumfangs in der Praxis)
- nur Integrationstest für Fremdsoftware, basierend auf (externer) Testspezifikation
- frühe Tests auf dem PC, auf Emulator, "Hilfs-Hardware" (z.B. aus Vorgängerprojekt)
- So viele Tests wie möglich automatisieren
- Nicht alle Tests lassen sich automatisieren
- Test der SW-Laufzeiten ist wichtig, inkl. Taskverhalten



Exkurs: Automotive Projekte und Software

Folgerungen:

- SW Test nicht im Detail über die ganze Laufzeit planen
 - Detailplanung nur für nächste Lieferung(en)
 - Grobe Planung über Gesamtprojekt
- Gesetzliche Anforderungen bei sicherheitskritischen Systemen, beachten von entsprechenden Normen (z.B. IEC 61508)
- Labortests sind wichtig (SW-, Komponenten- und System-Tests)
- Fahrzeugtests sind wichtig (kein SW-Test, sondern Systemtest)
- HW/SW Integrationstest ist wichtig



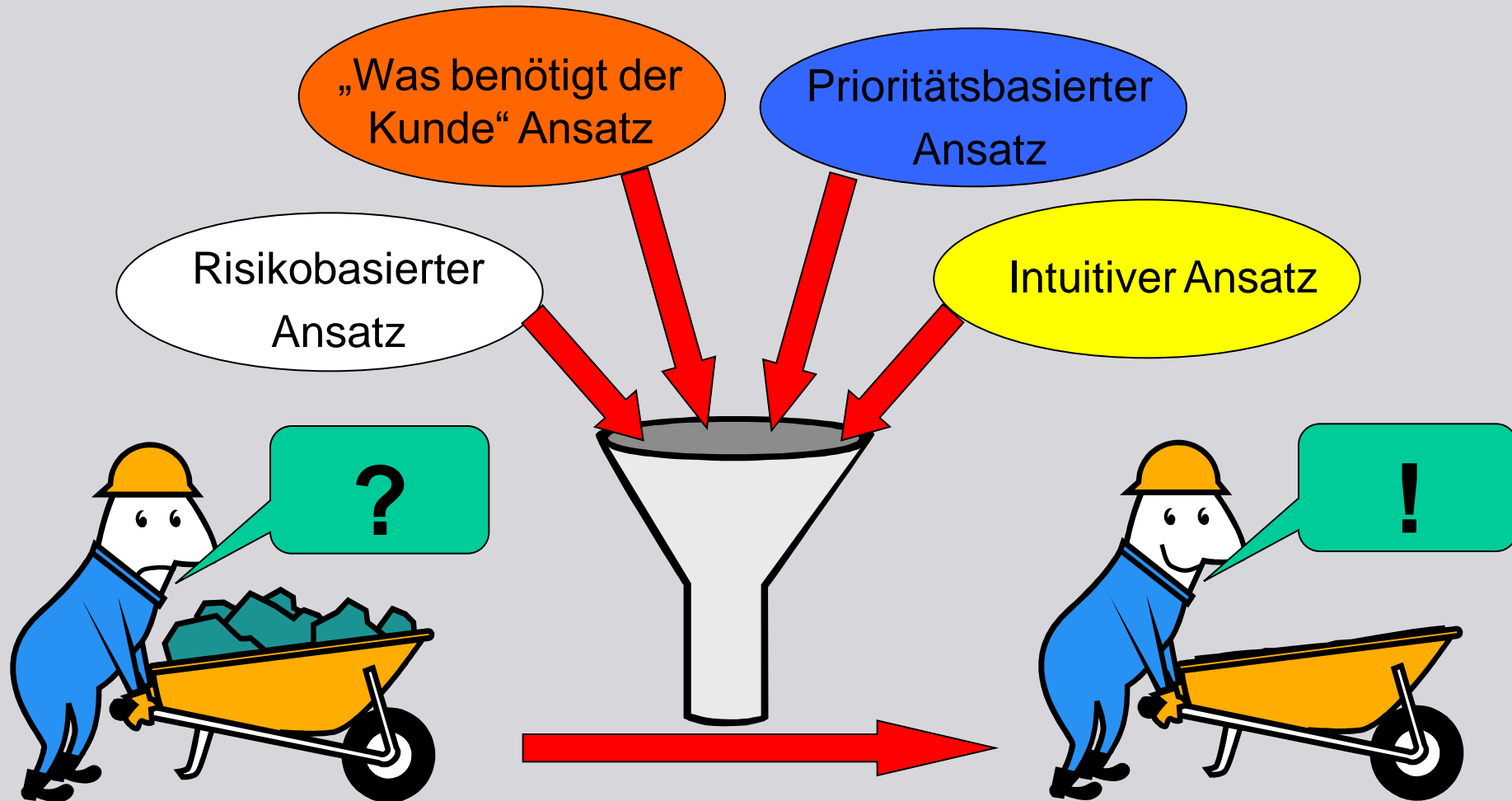
Zum Schluss dieses Abschnitts ...

Noch Fragen ??

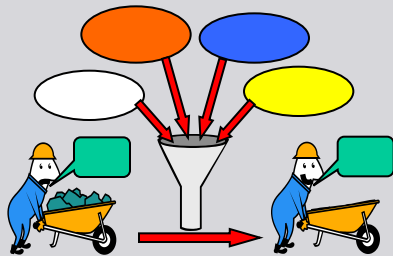
Inhalte

- 5.4.6 Weitere Begriffe – Regressions- und Abnahmetest
- 5.4.7 Test Dokumentation
- 5.4.8 Exkurs: Testen im Automotive Umfeld
- 5.4.9 Reduktion des Testumfangs**
- 5.4.10 Exkurs: Testen objektorientierter Software
- 5.4.11 Exkurs: Testdatengenerierung aus UML Modellen
- 5.4.12 Exkurs: Modellbasiertes Testen
- 5.4.13 Exkurs: Testen grafischer Bedienoberflächen
- 5.4.14 Do's und Dont's, Erfahrungen, Probleme

Reduktion des Testumfangs in der Praxis



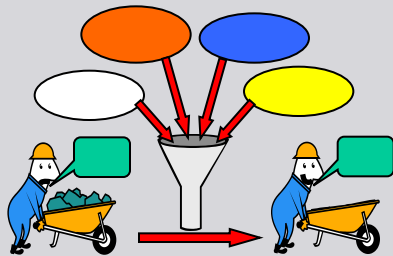
Reduktion des Testumfangs in der Praxis



„Was benötigt der Kunde“ Ansatz

- Nur das Testen, was der Kunde für seinen Test (z.B. Korrektur eines Fehlers, Labortest, Integration mit anderen Steuergeräten, Fahrzeugintegration) benötigt
- Für Prototypen, schnelle Fehlerbehebungen
- Manchmal schwierig umzusetzen, da nicht immer klar, was benötigt wird

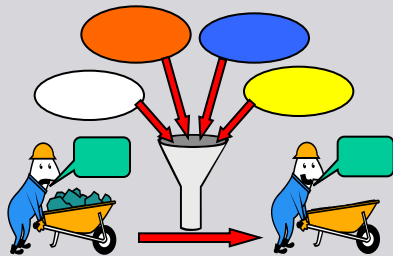
Reduktion des Testumfangs in der Praxis



Prioritätsbasierter Ansatz

- Testfälle priorisieren bzgl. Ihrer Wichtigkeit, z.B.
 - Prio 1: Testfälle für sicherheitskritische Funktionen
 - Prio 2: Testfälle, die die Gesamtfunktionalität des Systems aus Endkundensicht testen
 - Prio 3: Testfälle, die Einzelfunktionen testen
 - Prio 4: Testfälle der Funktionen, die dem Endkunden nicht sichtbar sind

Reduktion des Testumfangs in der Praxis



Risikobasierter Ansatz

- Funktionen in der SW bzgl. bestimmter Risikofaktoren bewerten, z.B.
 - Erfahrung der Entwickler mit dieser Funktion
 - Komplexität der Funktion
 - Bedeutung der Funktion (z.B. Sicherheitsrelevant)
 - Reife der Funktion (Übernahme oder Neuentwicklung)
- Gesamtrisiko (z.B. hoch, mittel, niedrig) einer Funktion aus ihren Risikofaktoren berechnen (mit geeigneter Gewichtung).
- Test der Funktion an das Gesamtrisiko der Funktion anpassen

Übung

- **Überlegen Sie eine Gewichtung beim Risikobasierten Test:**
- **Welche Risikofaktoren sind höher zu bewerten, welche niedriger?**
- **Welche Tests würden Sie reduzieren?**

10 min, arbeiten Sie ggf. zusammen mit einem Partner



Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

- 5.4.6 Weitere Begriffe – Regressions- und Abnahmetest
- 5.4.7 Test Dokumentation
- 5.4.8 Exkurs: Testen im Automotive Umfeld
- 5.4.9 Reduktion des Testumfangs
- 5.4.10 Exkurs: Testen objektorientierter Software**
- 5.4.11 Exkurs: Testdatengenerierung aus UML Modellen
- 5.4.12 Exkurs: Modellbasiertes Testen
- 5.4.13 Exkurs: Testen grafischer Bedienoberflächen
- 5.4.14 Do's und Dont's, Erfahrungen, Probleme

Übung

Testen objektorientierter Software



- Überlegen Sie, welche Eigenschaften / Features objektorientierter Software den Test beeinflussen können
- Überlegen Sie, welche Tests ggf. dazukommen, sich ändern, wegfallen, ...

15 min, arbeiten Sie ggf. zusammen mit einem Partner

Exkurs: Testen objektorientierter Software

Besonderheiten beim Test objektorientierter Software

- Test von Klassen
- Test von Klassenhierarchien (Vererbung)
- Testen virtueller Klassen (Klassen, bei denen nicht alle Methoden implementiert sind)
- Testen von Templateklassen (parametrisierte Klassen)
- Testen von Klassen- bzw. Objektinteraktionen
- Berücksichtigung der Besonderheiten der verwendeten objektorientierten Programmiersprache
 - C++
 - C#
 - Java



Exkurs: Testen objektorientierter Software

Vorgehen

- Prinzipiell lassen sich die bekannten, „herkömmlichen“ Teststrategien und Testtechniken auch auf Objektorientierte Software z.T. erweitert anwenden
- Weitere Testtechniken sind für Objektorientierte Software sinnvoll



Exkurs: Testen objektorientierter Software

Test von Klassen

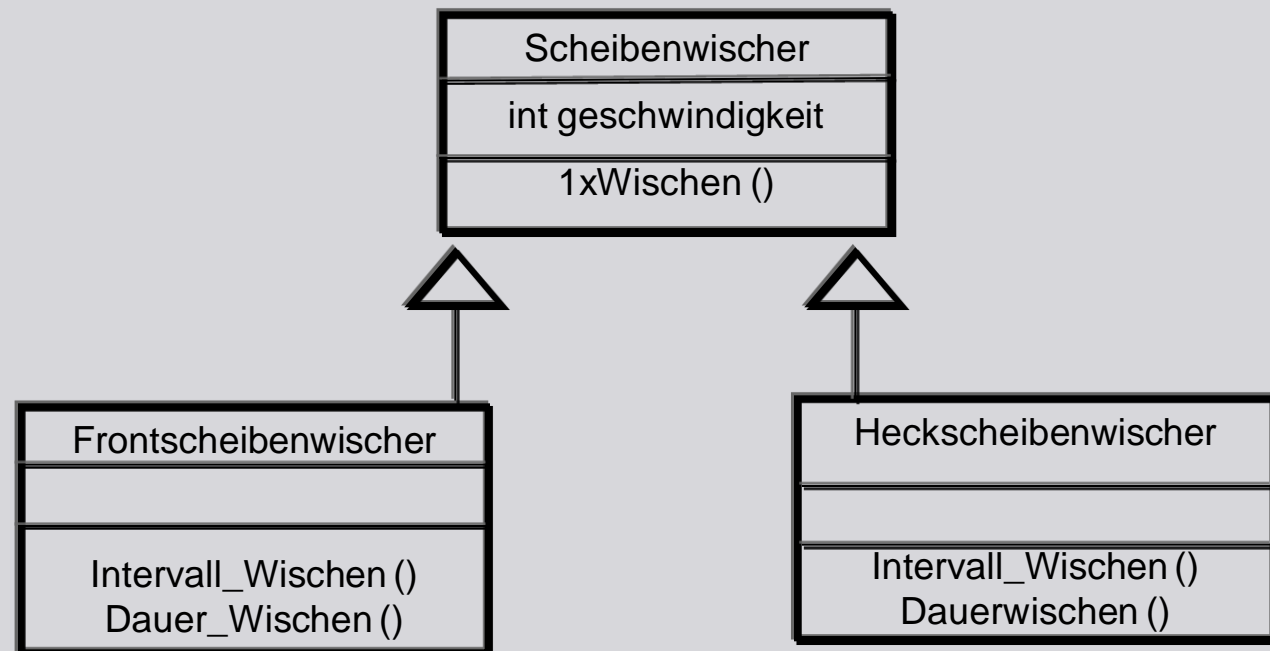
- Test einzelner Methoden (Member Functions)
 - Funktionaler Anteil
 - Test der Wirkung einer Methode auf die (gekapselten) Attribute des Aufrufobjekts
 - Berücksichtigung von Konstruktoren und Destruktoren
- Test des Zusammenspiels der Methoden einer Klasse
 - Integration der Methoden in die Klasse
 - Aufrufreihenfolgen testen (Konstruktor, Methode₁, Methode₂, ... Destruktor)



Exkurs: Testen objektorientierter Software

Test von Klassenhierarchien

- Test des Zusammenspiels der Methoden der Unterklasse mit den Methoden der Oberklassen
 - Integration der Methoden in die Klassenhierarchie
 - Test der Wirkung einer Methode auf die Oberklassenattribute
 - Aufrufreihenfolgen testen (Konstruktor, Methode₁, Methode₂, ... Destruktor)



Exkurs: Testen objektorientierter Software

Testen abstrakter Klassen

- Black-Box Testfälle definieren auf Basis der Spezifikation der virtuellen Methoden
- Test erweitern (z.B. um weitere Black Box und White Box Tests), wenn virtuelle Methoden in einer Unterklasse ausgeprägt (d.h. implementiert) werden

Test von Template Klassen

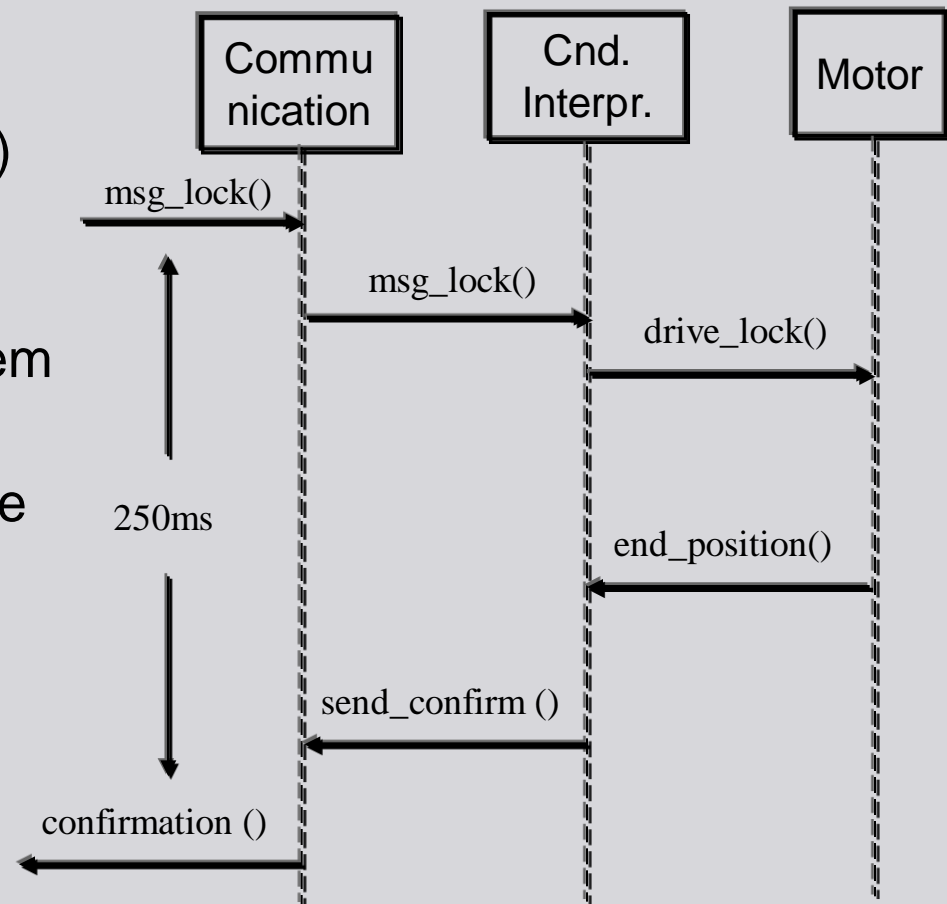
- Test von exemplarischen Ausprägungen der Templateklasse (Instantiierung mit einer entsprechenden Parameterklasse)
- Wiederverwendung dieser Testfälle bei neuen Instanziierungen
- Ergänzung um weitere Tests der Ausprägung



Exkurs: Testen objektorientierter Software

Testen der Objekt- bzw. Klassenkommunikation

- Sequence Charts als Basis
(Definition der Kommunikation zwischen Objekten bzw. Klassen)
- Test, ob die Objekte zur Laufzeit die spezifizierte Kommunikation durchführen, ggf. mit spezifiziertem Timing
- Test ggf. ergänzen durch negative Tests um Robustheit zu testen
(Objekt verhält sich nicht so, wie spezifiziert, wie reagiert die Umwelt)



Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

- 5.4.6 Weitere Begriffe – Regressions- und Abnahmetest
- 5.4.7 Test Dokumentation
- 5.4.8 Exkurs: Testen im Automotive Umfeld
- 5.4.9 Reduktion des Testumfangs
- 5.4.10 Exkurs: Testen objektorientierter Software
- 5.4.11 Exkurs: Testdatengenerierung aus UML Modellen**
- 5.4.12 Exkurs: Modellbasiertes Testen
- 5.4.13 Exkurs: Testen grafischer Bedienoberflächen
- 5.4.14 Do's und Dont's, Erfahrungen, Probleme

Exkurs: Testdatengenerierung aus UML Modellen

Vorbemerkungen

- Testdatengenerierung beinhaltet
 - Generierung von Testfällen
 - Generierung von Eingabedaten und Sollergebnissen von Testfällen
- Testdaten müssen u.U. manuell nachgearbeitet werden
- UML Diagramme werden meist verwendet, um Systeme bzw. Software objektorientiert zu modellieren.
 - ➔ Parallelen zum Test objektorientierter Software (Kap. 5.4.10)
- UML Diagramme werden zur modellbasierten Entwicklung genutzt.
 - ➔ Parallelen zum modellbasierten Test



Exkurs: Testdatengenerierung aus UML Modellen

Vorbemerkungen

- Bestimmte Diagramme (z.B. Zustandsdiagramme/State Diagrams) werden auch in anderen Modellierungssprachen bzw. Methoden verwendet.
- ➔ Methoden für UML können (ggf. angepasst) auch dort verwendet werden



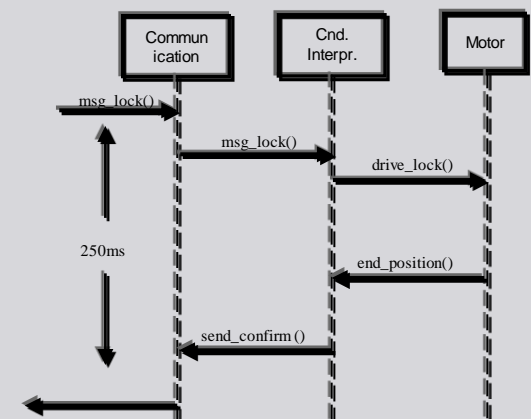
Exkurs: Testdatengenerierung aus UML Modellen

Zustandsdiagramme / State Diagrams

- Zustandsdiagramme beschreiben (vollständig) das dynamische Verhalten einer Modellelements mittels Zuständen, Zustandsübergängen und Ereignissen, die einen Übergang auslösen.
- Ziel: Testen, ob der implementierte Zustandsautomat der Spezifikation entspricht
- Testfälle der Form:
Ausgangszustand → Ereignis → Folgezustand
können direkt aus dem Zustandsdiagramm generiert werden.

Sequenzdiagramme / Sequence Charts

- s. Testen objektorientierter Software



Exkurs: Testdatengenerierung aus UML Modellen

Kommunikationsdiagramme / Communication Diagrams

- Beschreiben die Kommunikation zwischen Objekten bzw. Klassen
- Informationsgehalt ähnlich oder identisch zu Sequenzdiagrammen
- Ziel: Testen, ob die Kommunikation in der Implementierung der Spezifikation entspricht
- Testfälle können wie bei Sequenzdiagrammen abgeleitet bzw. generiert werden.

Aktivitätsdiagramme / Activity Charts

- Beschreiben interne Abläufe der Software
- Ziel: Testen, ob die Implementierung der Spezifikation entspricht (eine Art White-Box-Test)

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

- 5.4.6 Weitere Begriffe – Regressions- und Abnahmetest
- 5.4.7 Test Dokumentation
- 5.4.8 Exkurs: Testen im Automotive Umfeld
- 5.4.9 Reduktion des Testumfangs
- 5.4.10 Exkurs: Testen objektorientierter Software
- 5.4.11 Exkurs: Testdatengenerierung aus UML Modellen
- 5.4.12 Exkurs: Modellbasiertes Testen**
- 5.4.13 Exkurs: Testen grafischer Bedienoberflächen
- 5.4.14 Do's und Dont's, Erfahrungen, Probleme

Exkurs: Modellbasiertes Tests

Wesentliche Aspekte

- Basis
 - Statemate/Stateflow (Zustandsbasierte Systeme)
 - Matlab/Simulink (Regelungs-/Messtechnik)
 - UML Tools
- Statische Tests des Modells
 - gegen Modellierungsrichtlinien
 - auf Vollständigkeit und Konsistenz
 - durch Reviews
 - durch entsprechende Toolfeatures

Exkurs: Modellbasiertes Tests

Wesentliche Aspekte

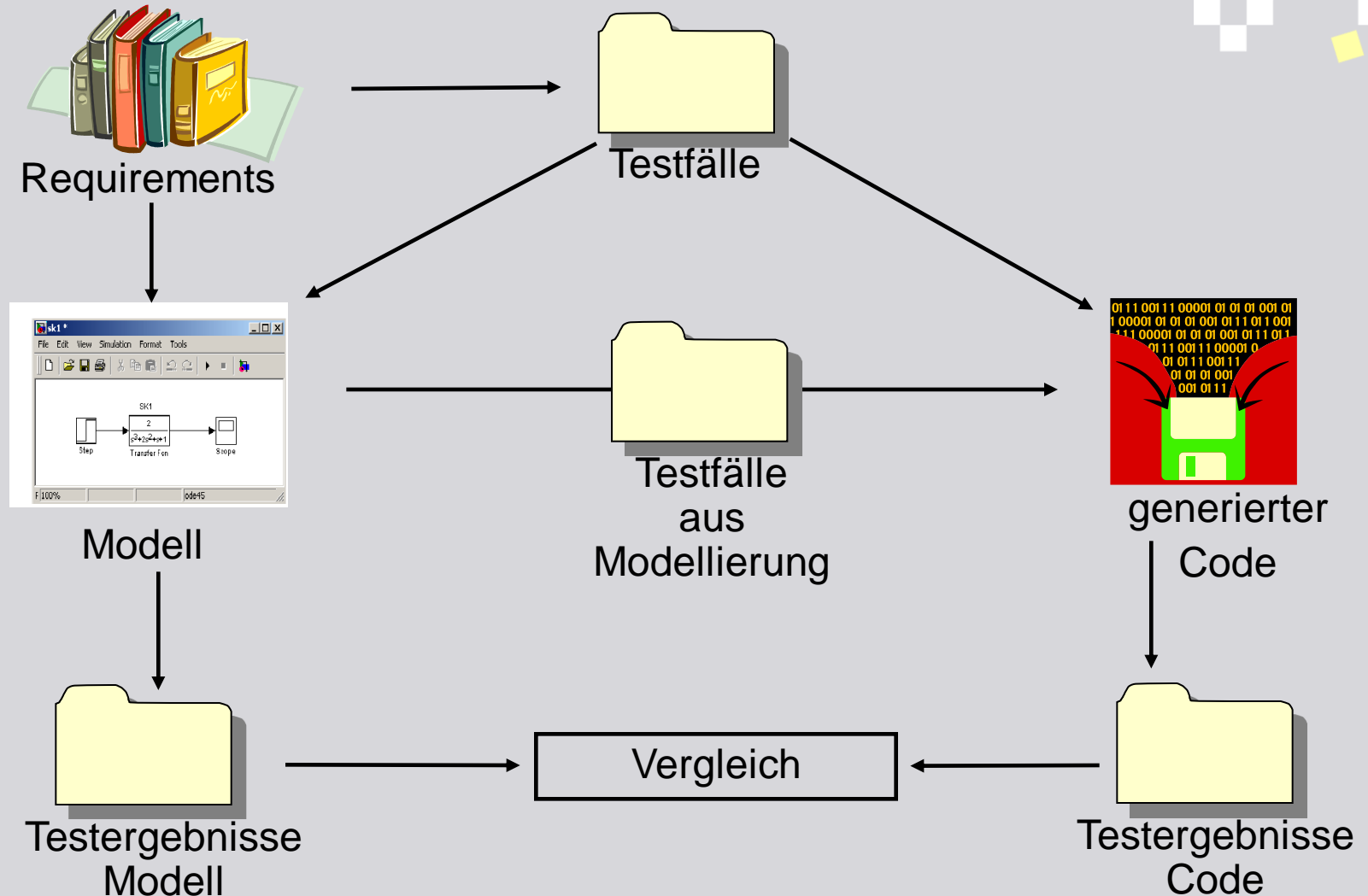
- Dynamischer Testen auf Modellebene (Simulation)
 - Test des Modells gegen die Requirements (Black-Box Test)
 - ➔ Test, ob das Modell die Requirements erfüllt
 - Modellabdeckung (White-Box Test)
 - ➔ Analog zur Codeüberdeckung, alle Modellelemente ausführen
- ➔ Voraussetzung für Modellverfeinerung (z.B. Fließkomma zu Festkommaarithmetik)
- ➔ Voraussetzung für Codegenerierung

Exkurs: Modellbasiertes Tests

Wesentliche Aspekte

- Test-/Simulationsdaten (-generierung) aus Modellen
 - zum Test auf Modellebene
 - zum Test von generiertem Code
 - zum Test von manuell erzeugtem Code
- ➔ Sicherstellen, dass sich der Code (soweit möglich) wie das Modell verhält, unter Berücksichtigung von
 - Verfeinerungen des Modells
 - verschiedenen Umgebungen (z.B. PC, Ziel-Hardware)
 - verschiedene Codegeneratoren

Exkurs: Modellbasiertes Tests



Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

- 5.4.6 Weitere Begriffe – Regressions- und Abnahmetest
- 5.4.7 Test Dokumentation
- 5.4.8 Exkurs: Testen im Automotive Umfeld
- 5.4.9 Reduktion des Testumfangs
- 5.4.10 Exkurs: Testen objektorientierter Software
- 5.4.11 Exkurs: Testdatengenerierung aus UML Modellen
- 5.4.12 Exkurs: Modellbasiertes Testen
- 5.4.13 Exkurs: Testen grafischer Bedienoberflächen**
- 5.4.14 Do's und Dont's, Erfahrungen, Probleme

Exkurs: Testen grafischer Bedienoberflächen

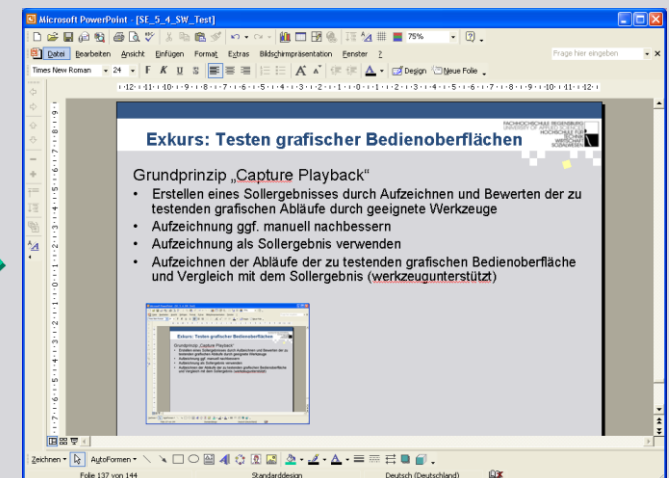
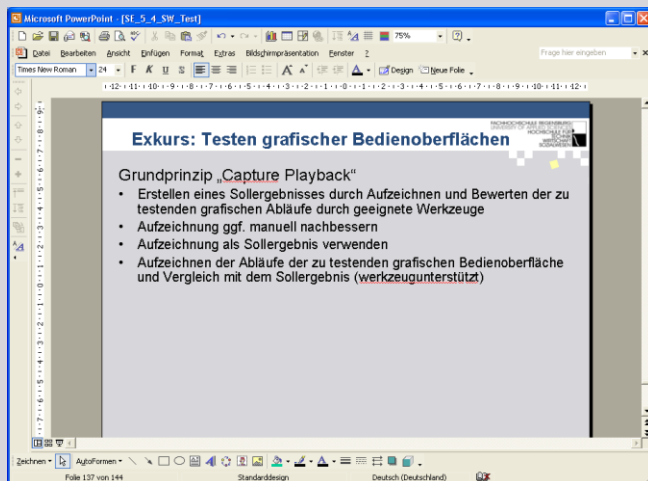
Vorbemerkungen

- Klassischer Testansatz (Test gegen Spezifikation) nicht 1-zu-1 übertragbar
- Verbale Spezifikation schwierig
- Übertragung der Spezifikation auf Testfälle schwierig
- Qualität einer Bedienoberfläche objektiv schwierig zu bewerten
- Nur möglich mit geeigneten Tools (z.B. WinRunner)

Exkurs: Testen grafischer Bedienoberflächen

Grundprinzip „Capture Playback“

- Erstellen eines Sollergebnisses durch Aufzeichnen und Bewerten der zu testenden grafischen Abläufe durch geeignete Werkzeuge
- Aufzeichnung ggf. manuell nachbessern
- Aufzeichnung als Sollergebnis verwenden
- Aufzeichnen der Abläufe der zu testenden grafischen Bedienoberfläche und Vergleich mit dem Sollergebnis (werkzeugunterstützt)
- Testtool führt Interaktionen (z.B. Mausklicks) wie bei Aufzeichnung durch



Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalte

- 5.4.6 Weitere Begriffe – Regressions- und Abnahmetest
- 5.4.7 Test Dokumentation
- 5.4.8 Exkurs: Testen im Automotive Umfeld
- 5.4.9 Reduktion des Testumfangs
- 5.4.10 Exkurs: Testen objektorientierter Software
- 5.4.11 Exkurs: Testdatengenerierung aus UML Modellen
- 5.4.12 Exkurs: Modellbasiertes Testen
- 5.4.13 Exkurs: Testen grafischer Bedienoberflächen
- 5.4.14 Do's und Dont's, Erfahrungen, Probleme**

"Do's" und "Don't"s, Erfahrungen, Probleme

- ☺ Module Test lohnt sich, auch im Embedded Umfeld
- ☺ Auch einfache Tests sollten dokumentiert werden
- ☺ SW Tests ergänzen Reviews und umgekehrt
- ☺ Tester möglichst unabhängig vom Rest der Entwicklung (ggf. für alle Testphasen)
- ☹ SW Modul Test ohne dokumentiertes SW Feindesign
- ☹ SW Integrationstest ohne dokumentierte SW Architektur
- ☹ SW Validierung ohne SW Requirements
- ☹ Erstellen der Testspezifikation wenn der Test beginnen soll
- ☹ Unterschätzen des Testaufwands ($\geq 50\%$ des Gesamtaufwands)
- ☹ geplante Testzeit als Zeitpuffer für andere Tätigkeiten missbraucht
- ☹ Überschätzen der Wirksamkeit von Test Tools

"Do's" und "Don't"s, Erfahrungen, Probleme

- ☹ Fehlende Traceability zu Requirements
- ☹ Nicht dokumentierte Tests
- ☹ Unerfahrene Mitarbeiter zum Testen „versetzt“ (Testen ist Kunst und Wissenschaft!)
- ☹ Testfälle von unklaren Anforderungen ableiten anstatt die Anforderungen zu klären
- ☹ Keine saubere Wartung von Testspezifikationen
- ☹ "Ich muss auf der Zielhardware testen" (Tests auf PC-Umgebungen sind möglich)
- ☹ Falscher Fokus/Ziel des Tests („meine SW ist schon korrekt“, destruktiver Modus wird nicht angewendet)
- ☹ Big Bang Integration
- ☹ SW Qualität kann nicht durch Test allein sichergestellt werden (Qualität in das Produkt hineintesten ist unmöglich!)

Zum Schluss dieses Abschnitts ...

Noch Fragen ??