

Bachelor of Science (BSc) in Informatik

Modul Advanced Software Engineering 2 (ASE2)

LE 06 – Software Testing

2 Grundlagen des Softwaretestens

Institut für Angewandte Informationstechnologie (InIT)

Walter Eich (eicw) / Matthias Bachmann (bacn)

<https://www.zhaw.ch/de/engineering/institute-zentren/init/>

Agenda

2 Grundlagen des Softwaretestens

2.1 Begriffe und Motivation

2.2 Softwarequalität

2.3 Der Testprozess

2.4 Die Psychologie des Testens

2.5 Wrap-up



Lernziele nach Syllabus CTFL (1/2)

1.1 Was ist Testen?

FL-1.1.1 (K1) Typische Ziele des Testens identifizieren können

FL-1.1.2 (K2) Testen von Debugging unterscheiden können

1.2 Warum ist Testen notwendig?

FL-1.2.1 (K2) Beispiele dafür geben können, warum Testen notwendig ist

FL-1.2.2 (K2) Die Beziehung zwischen Testen und Qualitätssicherung beschreiben können und Beispiele dafür geben können, wie Testen zu höherer Qualität beiträgt

FL-1.2.3 (K2) Zwischen Fehlhandlung, Fehlerzustand und Fehlerwirkung unterscheiden können

FL-1.2.4 (K2) Zwischen der Grundursache eines Fehlerzustands und seinen Auswirkungen unterscheiden können

1.3 Sieben Grundsätze des Testens

FL-1.3.1 (K2) Die sieben Grundsätze des Softwaretestens erklären können



Lernziele nach Syllabus CTFL (2/2)

1.4 Testprozess

- FL-1.4.1 (K2) Die Auswirkungen des Kontexts auf den Testprozess erklären können
- FL-1.4.2 (K2) Die Testaktivitäten und zugehörigen Aufgaben innerhalb des Testprozesses beschreiben können
- FL-1.4.3 (K2) Arbeitsergebnisse unterscheiden können, die den Testprozess unterstützen
- FL-1.4.4 (K2) Die Bedeutung der Pflege der Verfolgbarkeit zwischen Testbasis und Testarbeitsergebnissen erklären können

1.5 Die Psychologie des Testens

- FL-1.5.1 (K1) Die psychologischen Faktoren identifizieren können, die den Erfolg des Testens beeinflussen
- FL-1.5.2 (K2) Den Unterschied zwischen der für Testaktivitäten erforderlichen Denkweise und der für Entwicklungsaktivitäten erforderlichen Denkweise erklären können



2.1 Begriffe und Motivation

- **Testen**

- Der **Prozess**, der aus allen Aktivitäten des Lebenszyklus besteht (sowohl statisch als auch dynamisch), die sich mit der Planung, Vorbereitung und Bewertung eines Softwareprodukts und dazugehöriger Arbeitsergebnisse befassen. Ziel des Prozesses ist sicherzustellen, dass diese allen festgelegten Anforderungen genügen, dass sie ihren Zweck erfüllen, und etwaige Fehlerzustände zu finden.

Quelle: <http://glossar.german-testing-board.info/>, Stand 29.4.2019

2.1.1 Fehlerbegriff (1/3)

- **Fehlerwirkung**, Fehlfunktion, äusserer Fehler, Ausfall (engl. failure)
 - Ein Ereignis in welchem eine Komponente oder ein System eine geforderte Funktion nicht im spezifizierten Rahmen ausführt. [Nach ISO 24765]
- **Fehlerzustand** (engl. fault, defect, bug)
 - Defekt (innerer Fehlerzustand) in einer Komponente oder einem System, der eine geforderte Funktion des Produkts beeinträchtigen kann, z.B. inkorrekte Anweisung oder Datendefinition.
- **Fehlhandlung** (engl. error)
 - Die menschliche Handlung, die zu einem falschen Ergebnis führt. [Nach IEEE 610]



2.1.1 Fehlerbegriff (2/3)

Fehlhandlung
(error)



Fehlerzustand
(bug, defect)



Fehlerwirkung
(failure)



reduzierbar durch Schulung,
Prozessverbesserung

erkennbar in Peer-Reviews

final byte a = 5, b = 0;

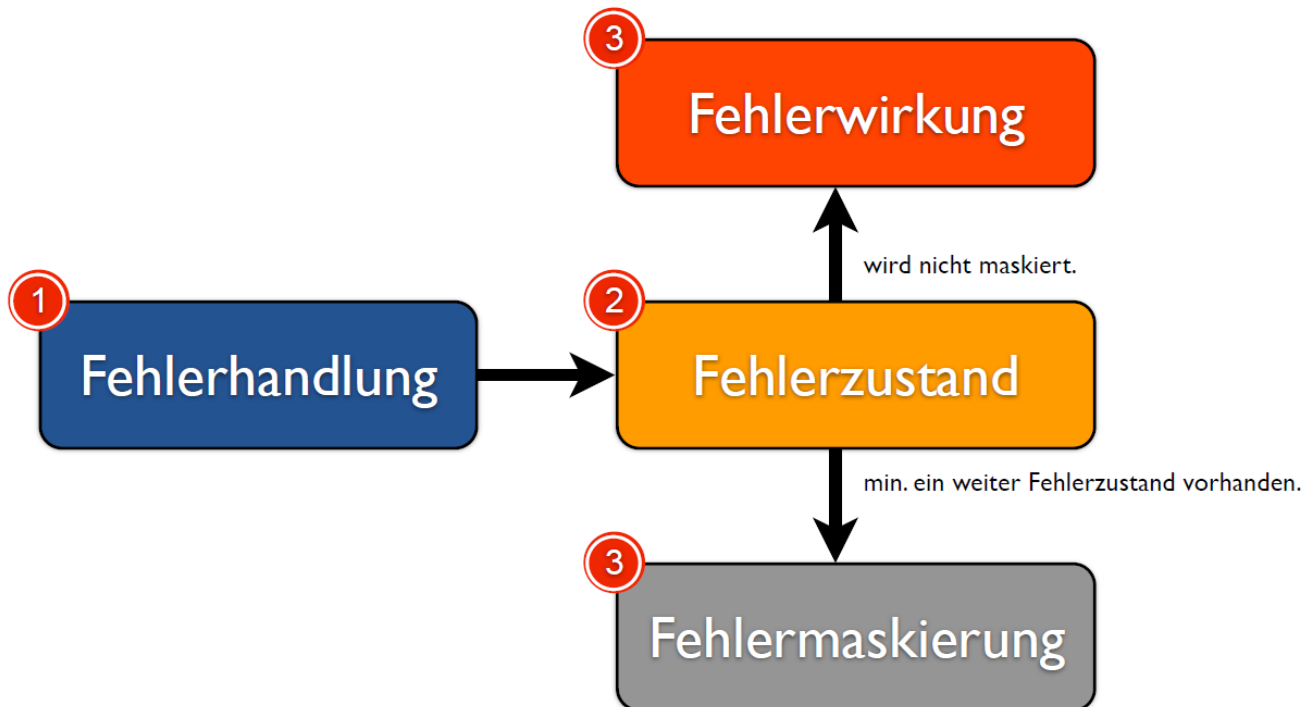
return a/b;

Nachweisbar durch Testen

```
C:>java Division
Exception in thread "main" java.lang.ArithmeticExc
eption: / by zero
    at Division.main(Division.java:7)
```

2.1.1 Fehlerbegriff (3/3)

- **Fehlermaskierung** (engl. defect masking)
 - Ein Umstand, bei dem ein Fehlerzustand die Aufdeckung eines anderen verhindert. [Nach IEEE 610]



Kurzer Einschub: Der erste Bug!

9/9


0800 Antan started
1000 " stopped - antan ✓

1300 (032) MP-MC ~~1.98264000~~ 2.130476415 (3) 4.615925059 (-2)
(033) PRO 2 2.130476415
conv 2.130676415

Relays 6-2 in 033 failed special speed test
in relay " 11.00 test.

Relays changed

1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.

1545  Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

1630 Antan started.
1700 closed down.

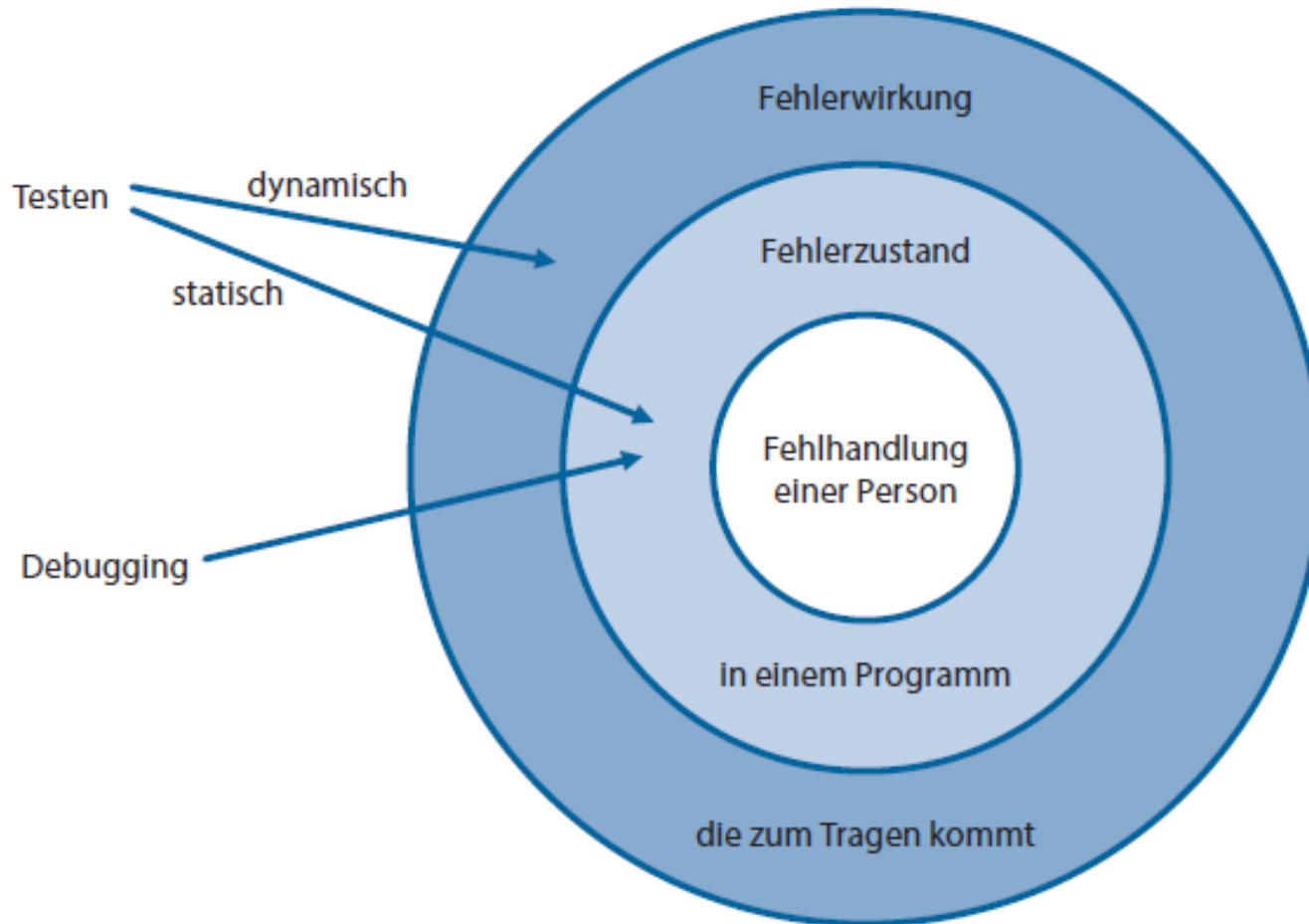
1.2700 9.037847025
9.037846895 conv
2.130476415 (3) 4.615925059 (-2)

Relay 2145
Relay 3376

Von Grace Hopper erzählt:
Logbuch-Seite des Mark II Aiken
Relay Calculator mit dem ersten *bug*
(1947)



Zusammenhang zwischen Fehlhandlung, Fehlerzustand und Fehlerwirkung



Falsch positives Ergebnis und falsch negatives Ergebnis (1/2)

- Nicht jedes unerwartete Ergebnis der Tests ist auch immer eine Fehlerwirkung.
 - Testergebnis zeigt Fehlerwirkung, obwohl der Fehlerzustand bzw. die Ursache für die Fehlerwirkung nicht im Testobjekt liegt
 - Dies wird als »falsch positives Ergebnis« («false-positive Result») bezeichnet.
 - Testergebnis zeigt keine Fehlerwirkung, obwohl die Tests diese hätten aufdecken sollen.
 - Dies wird als »falsch negatives Ergebnis« («false-negative Result») bezeichnet.
- Bei jeder Auswertung von Testergebnissen ist somit zu beachten, ob eine der beiden Möglichkeiten vorliegt.

Falsch positives Ergebnis und falsch negatives Ergebnis (2/2)

- Es gibt noch zwei weitere Ergebnisse:
 - «richtig positiv» (Fehlerwirkung durch den Testfall aufgedeckt) und
 - «richtig negativ» (erwartetes Verhalten bzw. Ergebnis des Testobjekts mit dem Testfall nachgewiesen).
- *Anmerkung: Diese Terminologie wird im Kapitel Testmanagement noch weiter ausgeführt.*

Aus Fehlern lernen

- Konnten Fehlerzustände aufgedeckt und die Fehlhandlungen ermittelt werden, dann lohnt es sich, mögliche **Ursachen zu analysieren**, um **daraus zu lernen** und in Zukunft gleiche oder ähnliche Fehlhandlungen zu vermeiden.
- Die so gewonnenen Erkenntnisse können zur **Prozessverbesserung** genutzt werden, um das Auftreten von zukünftigen Fehlhandlungen und damit Fehlerzuständen zu verringern oder zu verhindern.

2.1.2 Testbegriff (1/3)

- Wirkung eines Defekts
 - Um den Defekt zu korrigieren muss der Defekt lokalisiert werden.
 - Bekannt ist die Wirkung, aber nicht die genaue Stelle.
- Debugging (Fehlerbereinigung, Fehlerkorrektur)
 - Debugging und Testen sind verschiedene Dinge:
 - **Dynamische Tests** können Fehlerwirkungen zeigen, die durch Fehlerzustände verursacht werden.
 - **Debugging** ist eine Entwicklungsaktivität, die die Ursache (den Fehlerzustand) einer Fehlerwirkung identifiziert, analysiert und entfernt.

2.1.2 Testbegriff (2/3)

- Testen verfolgt mehrere Ziele:
 - **Qualitative Bewertung** von Arbeitsergebnissen wie Anforderungsspezifikation, User Stories, Design und Programmtext
 - Nachweis, dass alle **spezifischen Anforderungen vollständig umgesetzt** sind und dass das Testobjekt so funktioniert, wie es die Nutzer und andere Interessenvertreter (Stakeholder) erwarten
 - Informationen zur Verfügung stellen, damit die Stakeholder die Qualität des Testobjekts fundiert einschätzen können und somit **Vertrauen in die Qualität** des Testobjekts schaffen
 - **Höhe des Risikos** bei mangelnder Qualität der Software kann durch Aufdeckung (und Behebung) von Fehlerwirkungen verringert werden (weniger unentdeckte Fehlerzustände)

2.1.2 Testbegriff (3/3)

- Analysieren des Programms und der Dokumente, um **Fehlerzustände zu vermeiden** und vorhandene zu erkennen (und dann zu beheben)
- Analysieren und Ausführen des Programms mit dem Ziel, **Fehlerwirkungen nachzuweisen**
- **Informationen** über das Testobjekt zu erhalten, um zu entscheiden, ob das Systemteil für die Integration mit weiteren **Systemteilen freigegeben** (einchecken, »commit«) werden kann
- Aufzeigen, dass das Testobjekt **konform zu vertraglichen, rechtlichen oder regulatorischen Anforderungen** oder Standards ist und/oder Überprüfung der Einhaltung der Anforderungen oder Standards durch das Testobjekt

Unsystematischer Test

- **Laufversuch:** Der Entwickler „testet“
 - Entwickler übersetzt, bindet und startet sein Programm
 - Läuft das Programm nicht oder sind Ergebnisse offensichtlich falsch, werden die Defekte gesucht und behoben (**“Debugging”**)
 - Der „Test“ ist beendet, wenn das Programm läuft und die Ergebnisse vernünftig aussehen
- **Wegwerf-Test:** Testen ohne Systematik
 - Jemand „probiert“ das Programm mit verschiedenen Eingabedaten aus
 - Fallen „Ungereimtheiten“ auf, wird eine Notiz gemacht
 - Der Test endet, wenn der Tester findet, es sei genug getestet

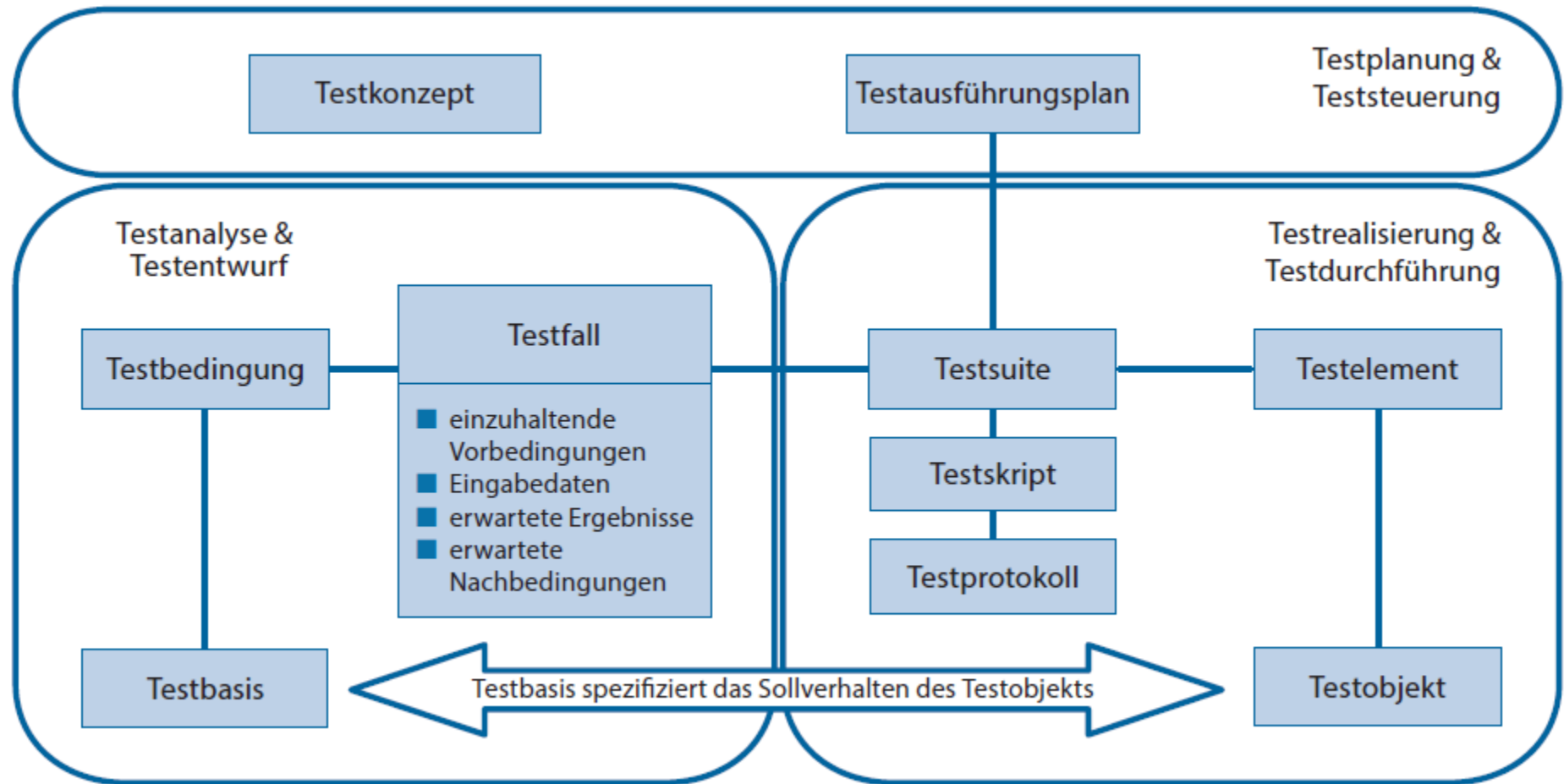
Ziele des systematischen Testens

- **Reproduzierbarkeit:** Ergebnisse entstehen nicht zufällig sondern systematisch und nachvollziehbar
- **Planbarkeit:** Aufwand und Nutzen (gefundene Fehler!) können prognostiziert werden
- **Wirtschaftlichkeit:** Aufwand und Nutzen werden optimiert: Testsuite-Optimierung, Dokumentation, Werkzeugeinsatz, Prozess-Optimierung
- **Risiko- und Haftungsreduktion:** Der systematische Test gibt Sicherheit, dass kritische Fehler nicht auftreten.

Systematischer Test

- Test ist **geplant**, eine Testvorschrift liegt vor.
- Das **Programm** wird **gemäss Testvorschrift** – der Testspezifikation – ausgeführt.
- **Ist-Resultate** werden mit **Soll-Resultaten** verglichen.
- **Testergebnisse** werden **dokumentiert**.
- **Fehlersuche** und **-behebung** erfolgen **separat** (in den höheren Teststufen).
- **Nicht bestandene Tests** werden **wiederholt**.
- **Test endet**, wenn vorher **definierte Testziele** erreicht sind.
- Die **Testspezifikation** wird **laufend aktualisiert**.

Testartefakte und Ihre Beziehungen



2.1.4 Aufwand für das Testen (1/4)

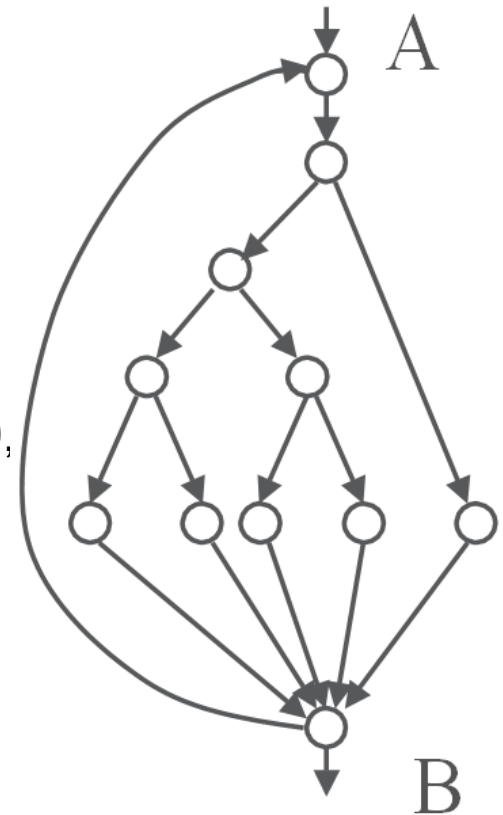
- Ein einfaches Programm soll getestet werden, das drei ganzzahlige Eingabewerte hat. Übrige Randbedingungen haben keinen Einfluss auf das Testobjekt.
 - Jeder Eingabewert kann bei 16 Bit Integer-Zahlen 2^{16} unterschiedliche Werte annehmen.
 - Bei drei unabhängigen Eingabewerten ergeben sich:
 $2^{16} * 2^{16} * 2^{16} = 2^{48}$ Kombinationen.
 - Jede dieser Kombinationen ist zu testen.
 - Wie lange dauert es bei 100.000 Tests pro Sekunde?

Es sind 281.474.976.710.656 Testfälle

Dauer: ca. 90 Jahre

2.1.4. Aufwand für das Testen (2/4)

- Ein einfaches Programm soll getestet werden, das aus vier Verzweigungen (IF-Anweisungen) und einer umfassenden Schleife besteht und somit fünf mögliche Wege im Schleifenrumpf enthält.
 - Unter der Annahme, dass die Verzweigungen voneinander unabhängig sind und bei einer Beschränkung der Schleifendurchläufe auf maximal 20, ergibt sich folgende Rechnung:
 - $5^{20} + 5^{19} + 5^{18} \dots + 5^1$
 - Wie lange dauert das Austesten bei 100.000 Tests pro Sekunde?



Es sind 119.209.289.550.780 Testfälle

Dauer: ca. 38 Jahre

2.1.4 Aufwand für das Testen (3/4)

- **Konklusion:** Ein Programm vollständig zu testen ist in der Praxis nicht möglich!
- Testaufwand in der Praxis:
 - 25% bis 50% des Entwicklungsaufwands
 - Testintensität und -umfang in Abhängigkeit vom Risiko und der Kritikalität festlegen
 - 2/3 des Testaufwandes können z.B. auf den Komponententest entfallen (agile Testpyramide)

2.1.4 Aufwand für das Testen (4/4)

- Testen unterliegt immer **beschränkten Ressourcen** (besonders Zeit, wenn Testen erst am Ende der Entwicklung in Angriff genommen wird)
- Besonders wichtig:
 - Adäquate (zum Testobjekt und den QS-Kriterien passende) Testverfahren auswählen
 - Unnötige Tests (die keine neuen Erkenntnisse liefern) vermeiden
 - Sowohl **Positiv-Tests** als auch **Negativ-Tests** berücksichtigen
 - Auch Tests auf Funktionalität, die nicht gefordert ist, können sinnvoll sein

2.1.5 Testwissen frühzeitig und damit erfolgreich nutzen

- **Beteiligen sich Tester an der Prüfung der Anforderungen** (z.B. durch Reviews) oder an der Verfeinerung («Refinements») von User Stories und bringen ihr Testfachwissen ein, können Unklarheiten und Fehler in den Arbeitsprodukten aufgedeckt und behoben werden.
- **Die enge Zusammenarbeit von Testern mit Systemdesignern** während des Entwurfs des Systems kann das Verständnis jeder Partei für das Design und dessen Test erheblich verbessern.
- **Arbeiten Entwickler und Tester während der Codeerstellung zusammen**, kann das Verständnis auf beiden Seiten für den Code und dessen Test verbessert werden.
- **Wenn Tester die Software vor deren Freigabe verifizieren und validieren**, können weitere Fehlerzustände erkannt und behoben werden, die ansonsten unentdeckt geblieben wären.

2.1.6 Grundsätze des Testens (1/3)

In den letzten 40 Jahren haben sich folgende Grundsätze zum Testen herauskristallisiert und können somit als Leitlinien dienen:

- Grundsatz 1: **Testen zeigt die Anwesenheit von Fehlerzuständen**
 - Mit Testen wird die Anwesenheit von Fehlerwirkungen nachgewiesen. Testen kann nicht zeigen, dass keine Fehlerzustände im Testobjekt vorhanden sind!
 - «Program testing can be used to show the presence of bugs, but never to show their absence!» Edsger W. Dijkstra, 1970
- Grundsatz 2: **Vollständiges Testen ist nicht möglich**
 - Vollständiges bzw. erschöpfendes Testen – Austesten – ist nicht möglich.
 - (s. Folien in Kapitel 2.1.4 zum Austesten)

2.1.6 Grundsätze des Testens (2/3)

- Grundsatz 3: **Frühes Testen spart Geld und Zeit**
 - Testen ist keine späte Phase in der Softwareentwicklung, es soll damit so früh wie möglich begonnen werden.
 - Durch frühzeitiges Prüfen (z.B. Reviews) parallel zu den konstruktiven Tätigkeiten werden Fehler(zustände) früher erkannt und somit Kosten gesenkt.
- Grundsatz 4: **Häufung von Fehlerzuständen**
 - Fehlerzustände sind in einem Testobjekt nicht gleichmässig verteilt, vielmehr treten sie gehäuft auf.
 - Dort wo viele Fehlerwirkungen nachgewiesen wurden, finden sich vermutlich auch noch weitere.

2.1.6 Grundsätze des Testens (3/3)

- Grundsatz 5: **Vorsicht vor dem Pestizid-Paradox**
 - Tests nur zu wiederholen, bringt keine neuen Erkenntnisse.
 - Testfälle sind zu prüfen, zu aktualisieren und zu modifizieren.
- Grundsatz 6: **Testen ist kontextabhängig**
 - Sicherheitskritische Systeme sind anders (intensiver, mit anderen Verfahren, ...) zu testen als beispielsweise der Internetauftritt einer Einrichtung.
- Grundsatz 7: **Trugschluss: „Keine Fehler“ bedeutet ein brauchbares System**
 - Ein System ohne Fehlerwirkungen bedeutet nicht, dass das System auch den Vorstellungen der späteren Nutzer entspricht.

2.2 Softwarequalität (Exkurs)

- Das **Testen von Software dient** durch die Identifizierung von Defekten und deren anschliessender Beseitigung zur **Steigerung der Softwarequalität**.
- Die **Testfälle sollen so gewählt werden**, dass sie weitgehend der **späteren Benutzung der Software entsprechen**.
- Die **nachgewiesene Qualität des Programms** während der Tests entspricht dann der zu **erwartenden Qualität während der späteren Benutzung**.
- **Softwarequalität umfasst aber mehr** als nur die Beseitigung der beim Test aufgedeckten Fehlerwirkungen (s. ISO-Norm 25010).

Software Produkt-Qualität nach DIN/ISO 25010



Qualitätsanforderungen

- **Qualitätsanforderungen** geben vor, welche **Qualitätsmerkmale** das Produkt in welcher **Güte** aufweisen soll (Qualitätsniveau).
 - Gesamtheit aller Qualitätsmerkmale und deren geforderte Ausprägung.
- Nicht alle Qualitätsmerkmale lassen sich gleich gut erfüllen.
 - Z.B. geht Effizienz oft zu Lasten der Wartbarkeit.
- Prioritäten festlegen
 - In engster Absprache mit Auftraggebern und Anwendern.
 - Qualitätsanforderungen sind Bestandteil der **nicht-funktionalen Anforderungen** im Lasten-/Pflichtenheft bzw. der Software Requirements Specification (SRS).

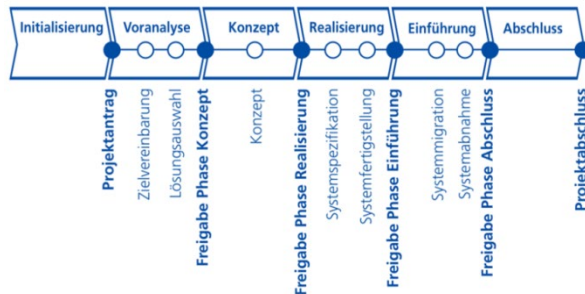
Testen und Qualität

- Testen misst die **Qualität** z.B. anhand der **Anzahl** gefundener **Fehlerwirkungen**.
- **Testen** erhöht **indirekt** die **Qualität**, da Fehler(zustände) vor der Auslieferung entdeckt werden.
- **Test** erhöht **indirekt** die **Prozessqualität**, da Fehler dokumentiert, analysiert und damit Fehlhandlungen in Zukunft vermieden werden können.
- **Testen** erhöht das **Vertrauen** in die **Qualität** des Systems, wenn wenige oder keine Fehler gefunden werden.

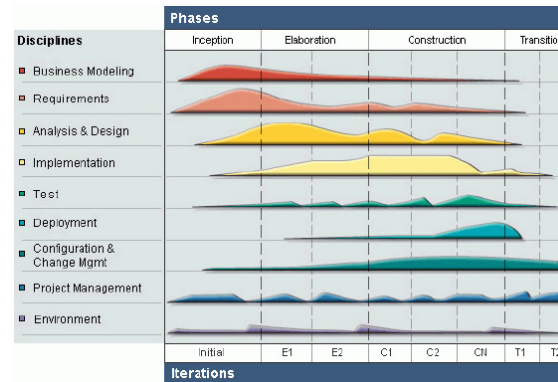
2.3 Der Testprozess (1/2)

Softwareprozessmodelle

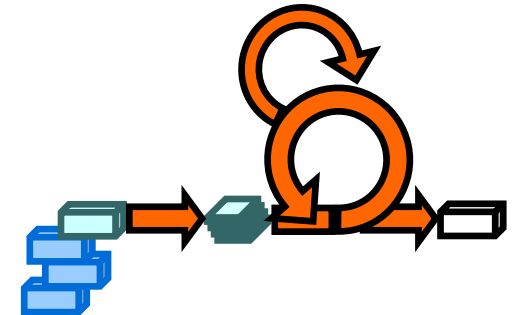
Hermes



RUP



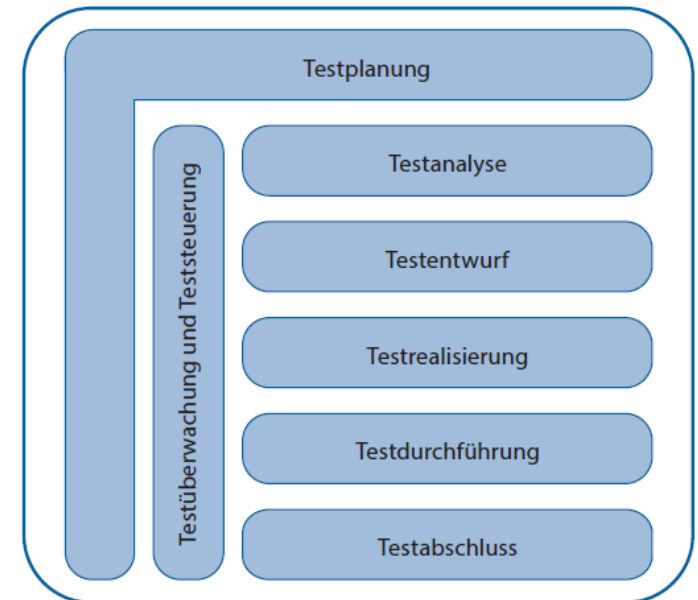
Scrum



- Das **Testen** findet sich **in jedem dieser Softwareprozessmodelle**, aber mit unterschiedlicher Bedeutung und Umfang.
- Grundlegende Testaktivitäten wie Testplanung, Testanalyse und -design lassen sich in einem **generischen Testprozess** beschreiben.

2.3 Der Testprozess (2/2)

- Ein **Testprozess** wird in der Regel folgende Aktivitäten umfassen (ISO-Norm 29119-2)
 - Testplanung
 - Testüberwachung und -steuerung
 - Testanalyse
 - Testentwurf
 - Testrealisierung
 - Testdurchführung
 - Testabschluss
- Diese **Aktivitäten** werden z.T. zeitlich **überlappend oder parallel** ausgeführt.
- Der Testprozess ist **für jede Teststufe** (Komponententest, Integrationstest, Systemtest und Abnahmetest) geeignet zu gestalten (Tailoring für ein Projekt).



2.3.1 Testplanung

- Die **Testplanung** einer so umfangreichen Aufgabe wie des Testens soll so früh wie möglich beginnen.
- **Aufgaben und Zielsetzung der Tests** müssen ebenso festgelegt werden wie die **benötigten Ressourcen**.
- Entsprechende Festlegungen sind im **Testkonzept** beschrieben:
 - Testziele
 - Teststrategie
 - Testaktivitäten
 - Ressourcen
 - Testbedingungen und Testbasis
 - Metriken
 - Risiken
- Die **Teststrategie** (auch Testvorgehensweise genannt) bildet den roten Faden.

2.3.2 Testüberwachung und Teststeuerung

- Die **Testüberwachung und Teststeuerung** umfasst
 - die fortwährende Beobachtung der aktuell durchgeführten Testaktivitäten im Vergleich zur Planung,
 - die Berichterstattung der ermittelten Abweichungen und die Durchführung der notwendigen Aktivitäten, um die geplanten Ziele auch unter den veränderten Situationen erreichen zu können.
- **Basis für Testüberwachung** und -steuerung sind **Endekriterien** für die jeweilige Testaktivität bzw. Testaufgabe.

2.2.3 Testanalyse (1/2)

Bei der **Testanalyse** geht es darum, zu ermitteln, was genau zu testen ist.

- **Testbasis prüfen:** Anforderungsspezifikationen, aus denen das Systemverhalten hervorgeht (z.B. Fachanforderungen, funktionale Anforderungen, Systemanforderungen, User Stories, Epics, Use Cases)
- **Dokumente analysieren:** Entwurfs- und Realisierungsinformationen, aus denen die Komponenten- oder Systemstruktur hervorgeht (z.B. System- oder Softwarearchitekturdokumente, Entwurfsspezifikationen, Aufrufdiagramme, UML- oder Entity-Relationship-Diagramme, Schnittstellenspezifikationen)
- **Testobjekt selbst prüfen:** Die Komponente oder das Systems selbst sind zu untersuchen - darunter Programmtext, Datenbank-Metadaten und -abfragen sowie weitere Schnittstellen
- **Berichte heranziehen:** Berichte zur Risikoanalyse, die funktionale, nicht funktionale und strukturelle Aspekte der Komponente oder des Systems beinhalten

2.2.3 Testanalyse (2/2)

- «Grundlage» für den gesamten Testprozess ist die **Testbasis**. Ist die Testbasis fehlerhaft, können keine «korrekten» Testbedingungen und damit keine «korrekten» Testfälle abgeleitet werden.
- Nachdem die Testbedingungen identifiziert und definiert sind, ist eine **Priorisierung der Testbedingungen** vorzunehmen. Damit soll sichergestellt werden, dass die wichtigen und risikoreichen Testbedingungen zuerst und ausreichend getestet werden.
- Bei der Testplanung soll sichergestellt werden, dass eine eindeutige **bidirektionale Rückverfolgbarkeit** (engl. traceability) zwischen der Testbasis und den anderen Arbeitsergebnissen der Testaktivitäten vorhanden ist.

2.3.4 Testentwurf

Beim **Testentwurf** geht es darum, festzulegen, wie getestet wird.

- Spezifikation von **abstrakten** und **konkreten Testfällen**
- **Identifizierung** benötigter **Testdaten**, um Definition von Testbedingungen und Testfällen zu unterstützen
- Für jeden Testfall ist die **Ausgangssituation (Vorbedingung)** beschrieben
- Klarheit, welche **Randbedingungen** für den Test gelten und einzuhalten sind
- Festlegung vor der Testdurchführung, welche **Ergebnisse bzw. welches Verhalten erwartet** wird
- Zur Bestimmung der **Sollergebnisse**, sind adäquate Quellen zu verwenden (→ Testorakel)

2.3.5 Testrealisierung

Aufgabe der **Testrealisierung** ist die abschliessende Vorbereitung aller notwendigen Aktivitäten, damit im nächsten Schritt die Testfälle zur Ausführung gebracht werden können.

- **Erstellung der Testmittel** (→ Testware), insbesondere ist die Testinfrastruktur im Detail zu realisieren
- **Testrahmen programmiert** und in der Testumgebung installiert
- **Abstrakte Testfälle sind zu konkretisiert**, d. h. mit den entsprechenden Testdaten versehen
- **Testfälle** sind zweckmässigerweise zu **Testsuiten gruppiert**, um die Testfälle während eines Testzyklus effektiv durchführen zu können und eine übersichtlich Struktur der Testfälle zu erhalten
- Testabläufe können oft direkt in **automatisierte Testskripts** überführt werden und verringern erheblich den zeitlichen Aufwand bei der Testdurchführung

2.2.6 Testdurchführung

Die **Testdurchführung** umfasst die konkrete Ausführung der Tests und deren Protokollierung.

- **Ausführung von Testabläufen** (manuell oder automatisiert) unter Einhaltung des Testplans (Reihenfolge, Testsuiten etc.)
- **Nachvollziehbarkeit** und Reproduzierbarkeit
- Vergleich der **Ist-Ergebnisse** mit den **vorausgesagten Ergebnissen** (→ Sollverhalten, Sollergebnis)
- gefundene **Fehlerwirkungen** oder **Abweichungen festhalten** und analysieren, um den Grund eines Problems festzustellen (z.B. Fehler im Code, in spezifizierten Testdaten, im Testdokument oder Fehler bei der Durchführung passiert)

2.2.7 Testabschluss (1/2)

- Beim **Testabschluss** – der letzten Aktivität im Testprozess – werden Daten aus den beendeten Testaktivitäten zusammengetragen, um **Erfahrungen auszuwerten** sowie Testmittel und weitere relevante Informationen zu konsolidieren.
- Für Ermittlung solcher **Metriken (oder Masse)** sind entsprechende Testwerkzeuge einzusetzen.
- Je nach Entwicklungsmodell gibt es unterschiedliche **Zeitpunkte** für den Testabschluss:
 - Freigabe eines Softwaresystems
 - Beendigung eines Testprojekts (oder auch dessen Abbruch)
 - Fertigstellung einer agilen Projektiteration
 - (z.B. als Teil einer Retrospektive/Bewertungssitzung)
 - Abschluss der Testaktivitäten auf einer Teststufe
 - Abschluss der Testaktivitäten zu einem Wartungsrelease

2.2.7 Testabschluss (2/2)

- Ein **Testabschlussbericht** ist zu erstellen.
- Er **fasst alle Testaktivitäten und -ergebnisse zusammen** und enthält eine abschliessende Bewertung der durchgeführten Tests gegen die festgelegten Endekriterien.
- Der Testabschlussbericht wird den **Stakeholdern zur Verfügung** gestellt.
- Die **Testmittel** (Testfälle, Testprotokolle, Testinfrastruktur, eingesetzte Werkzeuge usw.) sind zur weiteren Verwendung zu **archivieren**.

2.4 Psychologie des Testens (1/5)

- «Errare humanum est»
 - ... aber wer gibt es schon gerne zu?
- Entwicklung = konstruktiv
 - Test = destruktiv?
 - «Testen ist eine extrem kreative und intellektuell herausfordernde Aufgabe»

*Kann der Entwickler sein eigenes Programm testen?
Was meinen Sie?*

2.4 Psychologie des Testens (2/5)

- Blindheit gegenüber den eigenen Fehlern
 - Falls der Entwickler einen grundsätzlichen Designfehler eingebaut hat, z.B. weil er die Aufgabenstellung falsch verstanden hat, kann er das auch durch seine Tests nicht herausfinden.
 - Der passende Testfall wird ihm überhaupt nicht in den Sinn kommen.
- Kein Einarbeitungsaufwand
 - der Entwickler kennt sein Testobjekt

2.4 Psychologie des Testens (3/5)

- Unvoreingenommen
 - Es ist nicht «sein» Produkt, und mögliche Annahmen und Missverständnisse des Entwicklers sind nicht zwangsläufig auch die Annahmen und Missverständnisse des Testers.
- Einarbeitung notwendig
 - Um Testfälle zu erstellen, muss der Tester sich aber das benötigte Wissen über das Testobjekt aneignen, was Zeit kostet.
- Test-Know-how
 - bringt der Tester mit
 - ein Entwickler nicht bzw. er muss es sich erst aneignen (oder besser müsste, da dazu die notwendige Zeit oft nicht vorhanden ist).
 - ... bzw. er hat es auf der Hochschule gelernt

2.4 Psychologie des Testens (4/5)

- Stufen der **Unabhängigkeit von Tests** (von niedrig nach hoch):
 - vom Entwickler selbst
 - vom Kollegen des Entwicklers im gleichen Projekt
 - von Personen anderer Abteilungen
 - von Personen anderer Organisationen... durchgeführt.
- Bei allen Möglichkeiten können Werkzeuge eingesetzt werden.
- Die Aufteilung ist produkt- und projektabhängig.
- **Wichtig ist die richtige Mischung und Ausgewogenheit** zwischen «unabhängigen Tests» und Entwicklertests.

2.4 Psychologie des Testens (5/5)

- **Mitteilung von gefundenen Fehlerwirkungen** oder Abweichungen an den Entwickler oder/und dem Management verlangen
 - neutrale, objektive und konstruktive Art und Weise der Mitteilung und
 - ungestörte, offene Kommunikation.
- Anderen Menschen Fehlhandlungen nachzuweisen ist keine leichte Aufgabe, die viel »Fingerspitzengefühl« erfordert
- **Reproduzierbarkeit** ist wichtig!
 - Dokumentation der Testumgebung
 - Unterschiede zur Entwicklungsumgebung
- **Eindeutige Anforderungen**, präzise Spezifikation
 - «It's not a bug, it's a feature!»
- Förderlich für die Zusammenarbeit zwischen Tester und Entwickler ist die **gegenseitige Kenntnis der Aufgaben!**

Wrap-up

- Das **ISTQB Glossar** und **einschlägige Normen** definieren wichtige **Fachbegriffe** im Gebiet des **Softwaretests**.
- **Tests sind wichtige Massnahmen zur Qualitätssicherung** in der Softwareentwicklung.
- Das **Testen** nimmt einen **hohen Anteil am Entwicklungsaufwand** in Anspruch. Welcher Aufwand gerechtfertigt und zu investieren ist, hängt stark vom Charakter des Projekts ab.
- Die **sieben Grundsätze** des Testens sind stets zu beachten.
- Der **Testprozess** besteht aus den Aktivitäten Testplanung, Testüberwachung und Teststeuerung, Testanalyse, Testentwurf, Testrealisierung, Testdurchführung und Testabschluss.
- Menschen machen Fehler, aber sie geben es nur ungern zu! Aus diesem Grund spielen **beim Testen auch psychologische Aspekte** eine nicht zu vernachlässigende Rolle.

Ausblick

- Das Thema der nächsten Vorlesung ist:
 - Kap. 3 Testen im Softwareentwicklungslebenszyklus