Zürcher Hochschule
für Angewandte Wissenschaften

zh aw School of Engineering

Bachelor of Science (BSc) in Informatik

Modul Advanced Software Engineering 2 (ASE2)

LE 07 – Software Testing

3 Testen im Softwareentwicklungslebenszyklus

Institut für Angewandte Informationstechnologie (InIT)
Walter Eich (eicw) / Matthias Bachmann (bacn)
https://www.zhaw.ch/de/engineering/institute-zentren/init/

Agenda

Lernziele nach Syllabus ISTQB CTFL

2.1 Softwareentwicklungslebenszyklus-Modelle

FL-2.1.1 (K2) Die Beziehungen zwischen Softwareentwicklungsaktivitäten und Testaktivitäten im Softwareentwicklungslebenszyklus erklären können

FL-2.1.2 (K1) Gründe identifizieren können, warum Softwareentwicklungslebenszyklus-Modelle an den Kontext des Projekts und an die Produktmerkmale angepasst werden müssen

2.2 Teststufen

FL-2.2.1 (K2) Die unterschiedlichen Teststufen unter den Aspekten der Testziele, Testbasis, Testobjekte, typischen Fehlerzustände und Fehlerwirkungen sowie der Testvorgehensweise und Verantwortlichkeiten vergleichen können

2.3 Testarten

FL-2.3.1 (K2) Funktionale, nicht-funktionale und White-Box-Tests vergleichen können

FL-2.3.2 (K1) Erkennen können, dass funktionale, nicht-funktionale und White-Box-Tests auf jeder Teststufe eingesetzt werden können

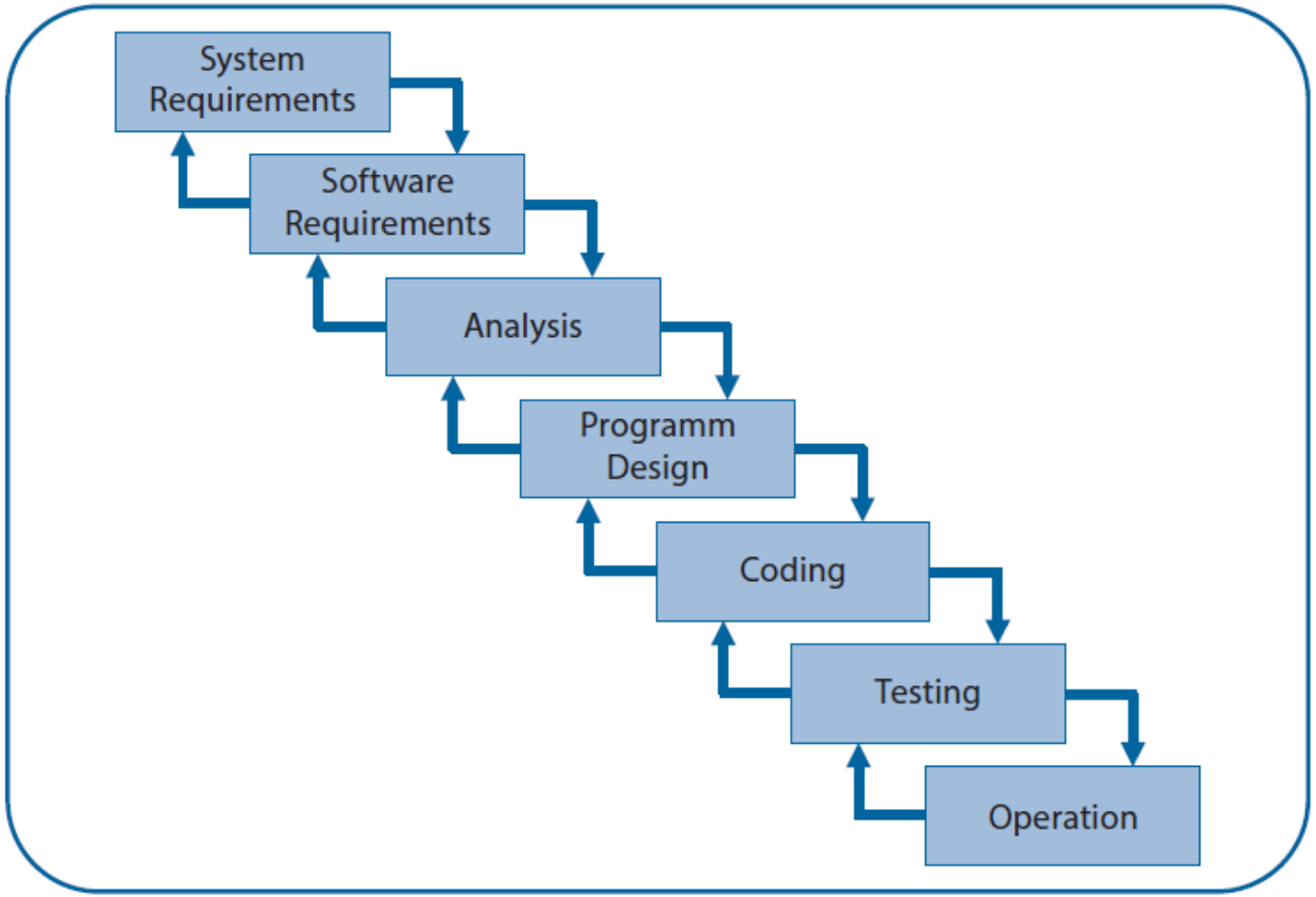
FL-2.3.3 (K2) Den Zweck von Fehlernachtests und Regressionstests vergleichen können

2.4 Wartungstest

FL-2.4.1 (K2) Auslöser für Wartungstests zusammenfassen können

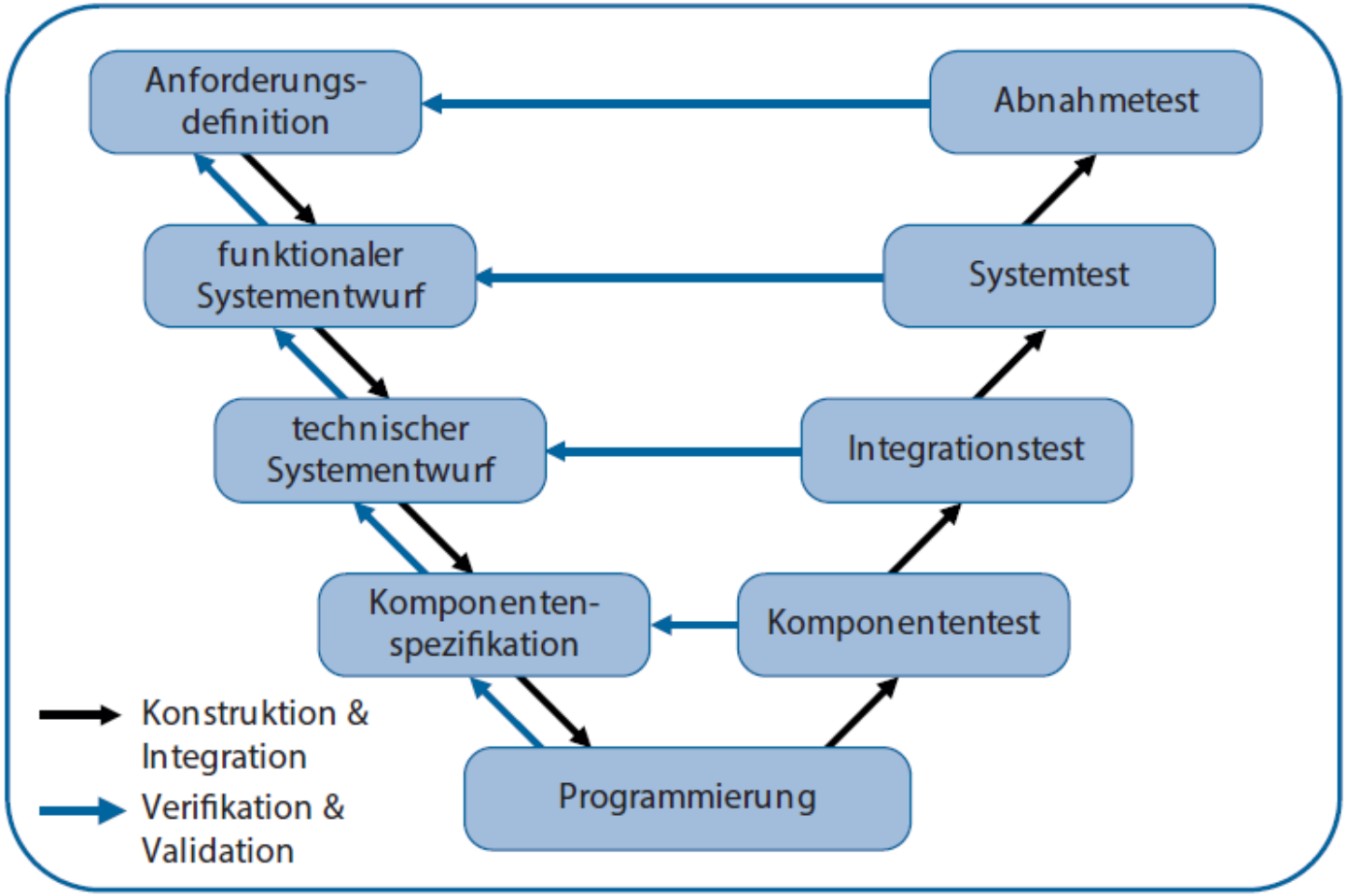
FL-2.4.2 (K2) Den Einsatz der Auswirkungsanalyse im Wartungstest beschreiben können

3.1.1 Das Wasserfallmodell



[Royce 70]

3.1.2 Das Allgemeine V-Modell (1/3)

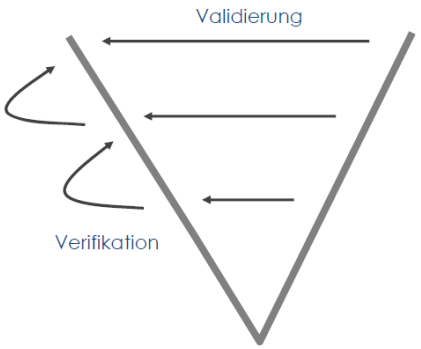


[Böhm 79]

3.1.2 Das Allgemeine V-Modell (2/3)

Sprachgebrauch im V-Modell nach ISTQB

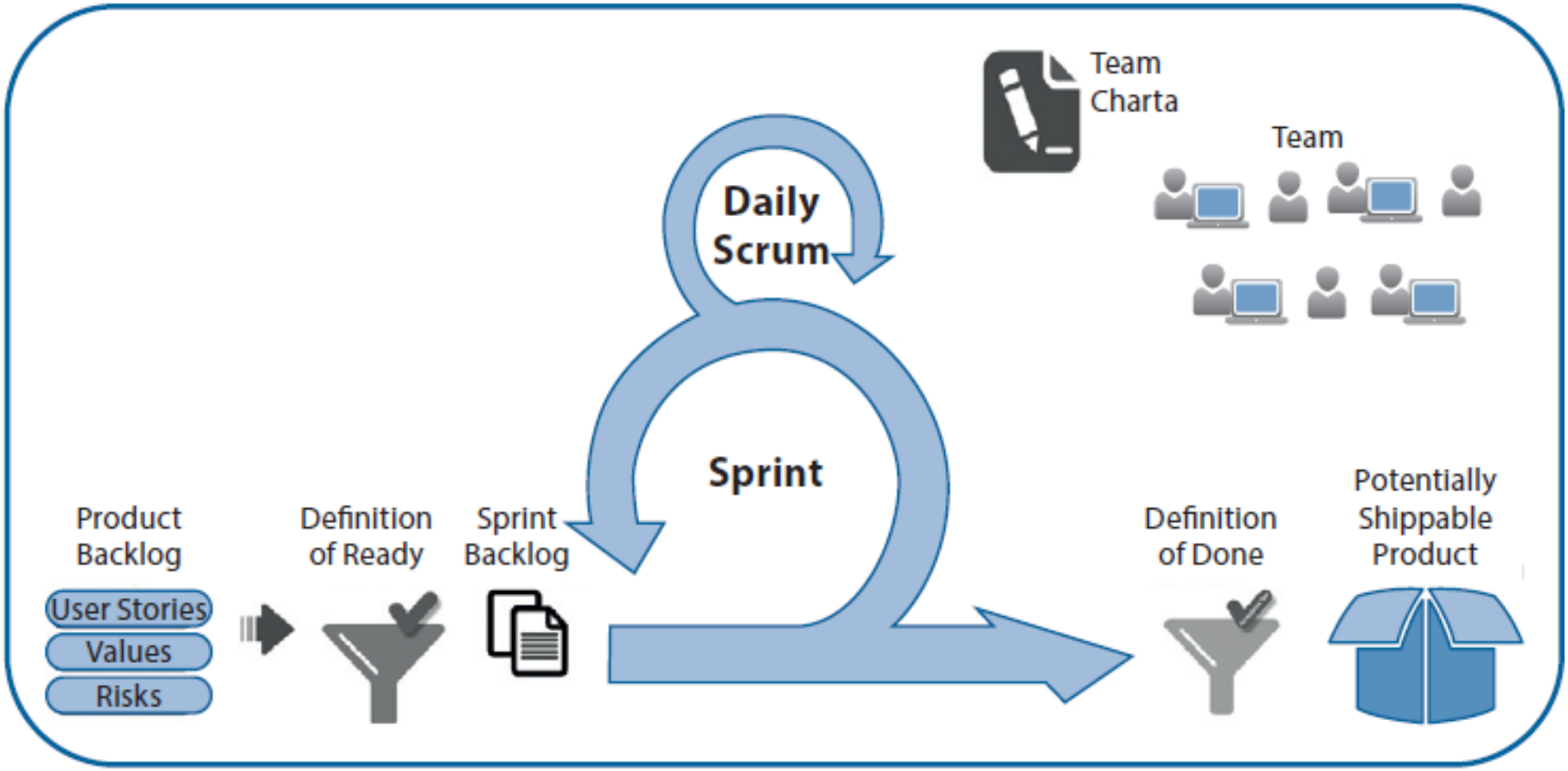
- **Verifikation**: Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind. [ISO 9000] („Are we doing the thing right?“)
- **Validierung**: Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind. [ISO 9000] („Are we doing the right thing?“)
- **Alternativer Sprachgebrauch**
  - Verifikation = formaler Korrektheitsbeweis
  - Validierung = informelle Überprüfung
- In der Praxis beinhaltet jeder Test beide Aspekte, wobei der validierende Teil mit steigender Teststufe zunimmt!



3.1.2 Das Allgemeine V-Modell (3/3)

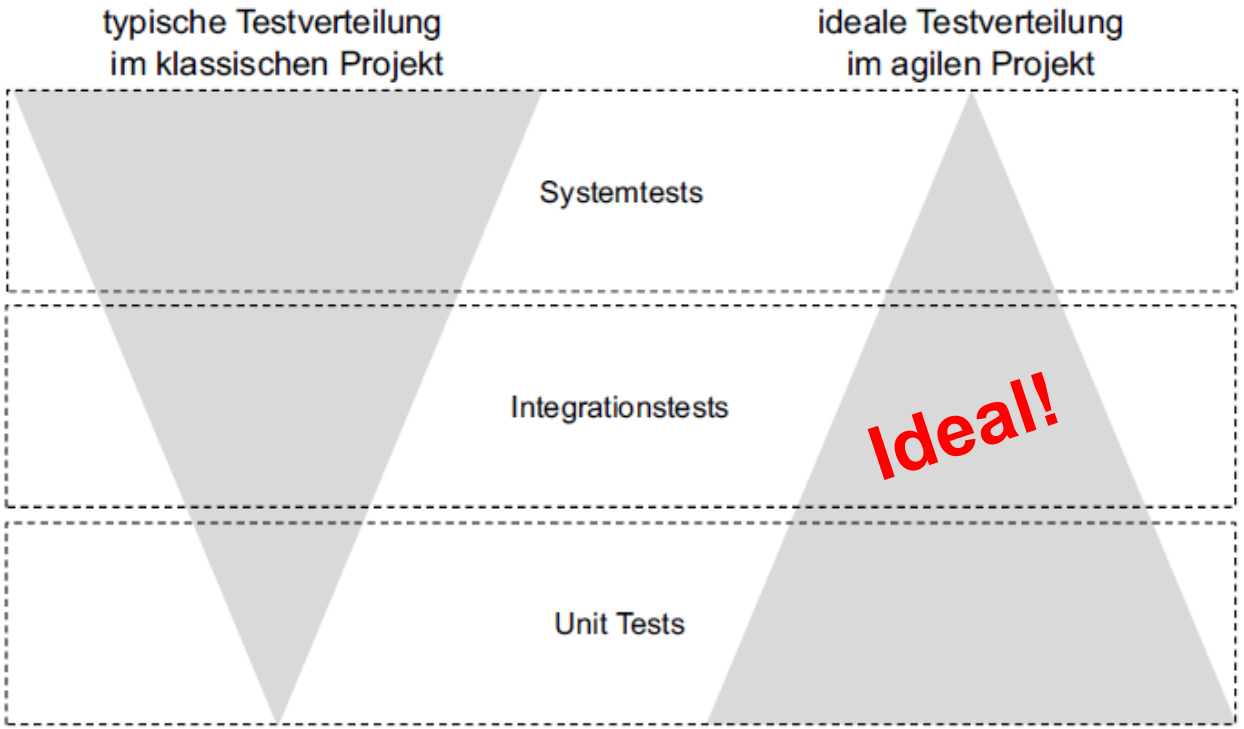
- Modellvorstellungen des V-Modells
  - Konstruktions- und Testaktivitäten sind getrennt, aber gleichwertig (linke Seite/rechte Seite).
  - Das «V» veranschaulicht die Prüfaspekte Verifizierung und Validierung.
  - Es werden arbeitsteilige Teststufen unterschieden, wobei jede Stufe «gegen» ihre korrespondierende Entwicklungsstufe testet.
- Testen innerhalb des Softwareentwicklungslebenszyklus
  - Analyse und Entwurf der Tests während der Entwicklungsaktivität
  - Tester sollen im Review Prozess für Requirements oder Architektur als Stakeholder miteingebunden werden

3.2 Iterative und inkrementelle Entwicklungsmodelle



Testpyramide im klassischen vs. agilem Projekt

- Die Testpyramide nach Cohn ist ein Konzept bzw. eine Metapher, die dem Scrum-Team hilft, zu überprüfen, ob die vorhandenen Testfälle angemessen über sämtliche Teststufen verteilt sind.



Testmanagement in Scrum (1/4)

- Ein Scrum-Team ist ein sich selbst organisierendes, interdisziplinäres Team.
- Es gibt keinen Projektleiter, der dem Team sagt, was zu tun ist, sondern Scrum vertraut darauf, dass ein Team sich selbst steuert.
- Das Team ist gemeinsam für alle Arbeiten zuständig.
- Programmierung und Testen sind nicht getrennt, sondern werden im selben Team gemeinsam erledigt.
- Dementsprechend gibt es in einem Scrum-Team auch keinen «Teilprojektleiter Test».
- Die bisher genannten Testmanagementaufgaben existieren natürlich dennoch und müssen anders verteilt werden.
- Die organisatorischen Aufgaben («Den Test organisieren») fallen in den Aufgabenbereich des Scrum Masters.

Testmanagement in Scrum (2/4)

- Die Leitungsaufgaben werden im Rahmen der Sprint-Planungspraktiken abgedeckt.
- So werden Testaufgaben entweder explizit über eigene Tasks geplant und überwacht oder implizit als Teil der Done-Kriterien anderer Aufgaben.
- Die Erfassung und Auswertung von Testfortschritt und Testergebnissen erfolgen bei funktionierender «Continuous Integration» hoch automatisiert, sodass manuelle Aufgaben in diesem Bereich minimiert sind.
- Continous Integration kann um den Schritt «Continuous Deployment» (CD) erweitert werden: Wenn die Tests fehlerfrei ablaufen, wird das getestete System automatisiert in die Produktionsumgebung kopiert und dort installiert (Deployment).
- Der «klassische Testmanager» wird hier in der Tat weitgehend überflüssig.
- Anders sieht es bei den testfachlichen Aufgabenbereichen «Teststrategie festlegen» und «Test inhaltlich planen» aus.

Testmanagement in Scrum (3/4)

- Scrum erwartet theoretisch, dass alle diese mit Testen zusammenhängenden Sachentscheidungen vom Team gemeinsam vorbereitet und getroffen werden.
- Wie in anderen Fachgebieten, so sind aber auch im Softwaretest ein ausreichendes Know-how und eine gewisse Erfahrung im Fachgebiet eine Voraussetzung für sachgerechte Entscheidungen.
- Ein Teammitglied mit Testexpertise wird benötigt.
- Deshalb sollte mindestens eine Person im Team «hauptamtlich» für Testen zuständig sein und über eine Ausbildung und Erfahrung als professioneller Softwaretester verfügen.
- Diese Person steuert dann ihre spezielle Expertise bei, um die Softwaretests inhaltlich fachgerecht, risikoorientiert und wirtschaftlich aufzusetzen und über alle Sprints hinweg zu realisieren, und sie berät den Product Owner bezüglich Produktqualität und Produktfreigaben.

Testmanagement in Scrum (4/4)

- Es spricht nichts dagegen, diese Person auch im Scrum-Team «Testmanager» zu nennen.
- Einen guten Beitrag können auch externe Testspezialisten leisten, die zur methodischen Unterstützung angefordert und eingesetzt werden.
- Auch in agilen Projekten sind die grundlegenden, klassischen Test-(design)techniken, wie Äquivalenzklassenanalyse, Grenzwertanalyse oder zustandsbasierter Test, unverzichtbar.
- Alle Teammitglieder, also auch die Softwareentwickler, sollten darin geschult werden.
- Der Scrum Master oder das in der Rolle des Testmanagers tätige Teammitglied muss dafür sorgen, dass diese klassischen Techniken angewendet werden.

3.3 Softwareentwicklung im Projekt- und Produktkontext

- Die Anforderungen an Planung und Nachvollziehbarkeit von Entwicklung und Test sind in unterschiedlichen Kontexten verschieden.
- Folgende Projekt- und Produktmerkmale können eine Rolle spielen:
  - Die Geschäftsprioritäten, Projektziele und Projektrisiken des Unternehmens, z.B. Time to Market als primäre Anforderung
  - Die Art des zu entwickelnden Produkts
  - Das Marktumfeld und technische Umfeld, in dem das Produkt eingesetzt wird, z.B. kleines vs. grosses System
  - Identifizierte Produktrisiken, z.B. Risiken eines sicherheitskritischen Systems (z.B. bei einem Fahrzeug-Bremssteuerungssystem)
  - Organisatorische und kulturelle Aspekte, z.B. verteiltes Team
- Für den Einsatz in einem konkreten Projekt kann und soll ein Vorgehensmodell auf die projektspezifischen Gegebenheiten angepasst und zugeschnitten werden («Tailoring»).

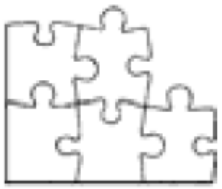
3.4 Teststufen

- Beim Testen kann und muss das zu testende System, seine Eigenschaften und sein Verhalten auch auf den verschiedenen Ebenen der Architektur, von den elementaren Einzelkomponenten bis zum Gesamtsystem, betrachtet und geprüft werden.
- Die Testaktivitäten einer solchen Ebene werden dabei als «Teststufe» bezeichnet.
- Jede Teststufe ist eine Instanz des Testprozesses.
- Die folgenden Folien erklären, worin sich das Testen auf den verschiedenen Teststufen hinsichtlich Testobjekt, Testzielen, Testmethoden und Verantwortlichkeiten unterscheidet.

Übersicht Teststufen

- Testgegenstand sind Komponenten, teilintegrierte Systeme und das Gesamtsystem

- **3.4.1 Komponententest**, Modultest (Unit-Test)
  - «autarke» Komponenten werden isoliert getestet
- **3.4.2 Integrationstest**
  - Teilintegrierte Komponenten testen die Komponenteninteraktion
- **3.4.3 Systemtest**
  - Test des Gesamtsystems
- **3.4.4 Abnahmetest**
  - Abnahme des Gesamt- oder Teilsystems durch Auftraggeber



3.4.1 Komponententest

«Testen im Kleinen» (Liggesmeyer)

- Das Testobjekt beim Komponententest ist eine Komponente (= Unit, Modul, Subsystem).
  - Komponenten sind ein Teil einer Applikation, z. B. eine (oder mehrere) Klassen oder eine Prozedur.
  - Es gibt weder feste Vorgaben für die Grösse einer Komponente (z.B. in LOC) oder für die Anzahl der Komponenten im Gesamtsystem.
  - In der Praxis findet man Systeme, die aus 10 Komponenten aufgebaut sind bis hin zu Systemen, die aus 10’000 Komponenten und mehr aufgebaut sind.
- Im Komponententest wird die Komponente an den Schnittstellen gegen die Spezifikation und das Softwaredesign getestet (Testbasis). Dazu ist eine Komponenten-Spezifikation erforderlich!

3.4.1 Komponententest - Begriffserklärung

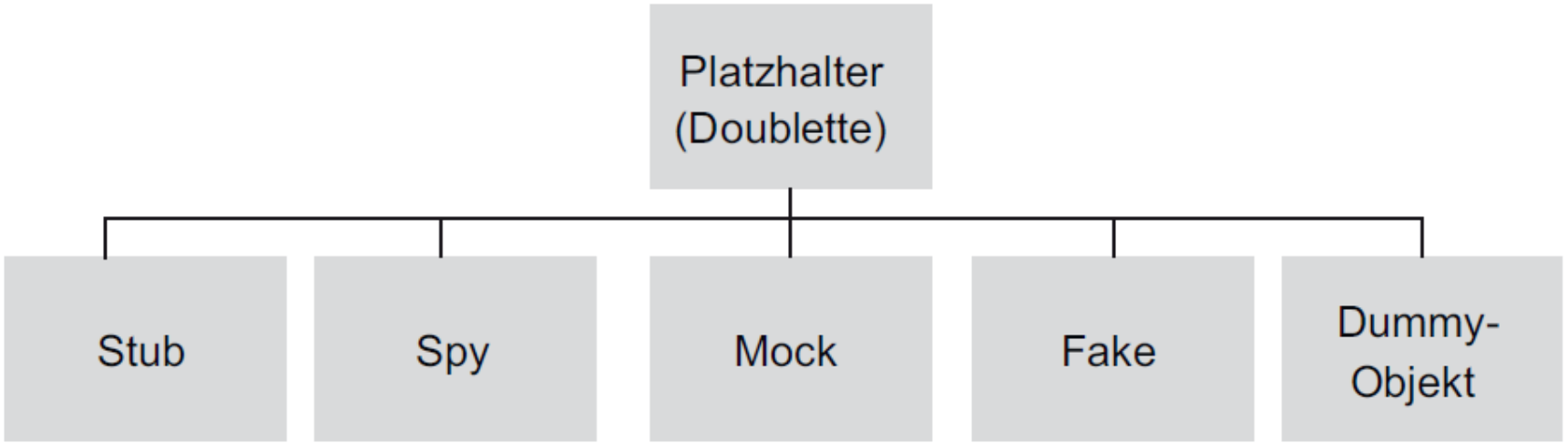
- **Testbasis:**
  - Anforderungen an die Komponente (Komponentenspezifikation)
  - detaillierter Softwaredesign
  - Code

- **Typische Testobjekte:**
  - Komponenten, Klassen(-verbund)
  - Programme
  - Datenumwandlung/Migrationsprogramme
  - Datenbankmodule

3.4.1 Komponententest - Testobjekte

- Die Komponente sollte möglichst isoliert getestet werden; die Isolierung hat dabei den Zweck, komponentenexterne Einflüsse beim Test auszuschliessen.
- Die Schnittstelle einer Komponente ist in der Regel eine Programmierschnittstelle. D.h., die Ausführung der Testfälle wird in der Regel in der Programmiersprache des Moduls programmiert.
- Wird ein Defekt gefunden, lässt sich die Ursache in der Regel der getesteten Komponente zuordnen.
- Es stehen Frameworks zur Verfügung: www.junit.org, Visual Studio 20xx, www.testng.org, CppUnit, TESSY, Rational Test Realtime, …

Stubs, Mocks und Dummies in agilen Projekten (Exkurs)



- Bei der Aufwandsplanung muss sich das Team darüber im Klaren sein, dass im Scrum-Projekt die Nutzung von Testplatzhaltern von höherer Bedeutung ist als in klassischen Projekten!

Testumgebung – JUnit / mockito (Exkurs)

- JUnit 4.x/5.x ist ein Komponententest-Framework für Java
- Mockito Mocking-Framework für java


```java
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class FirstJUnit5Tests {

    @Test
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }

}
```

```java
import static org.mockito.Mockito.*;

// mock creation
List mockedList = mock(List.class);

// using mock object - it does not throw any "unexpected interaction" exception
mockedList.add("one");
mockedList.clear();

// selective, explicit, highly readable verification
verify(mockedList).add("one");
verify(mockedList).clear();
```

3.4.1 Testziele Komponententest

- Test der Funktionalität
  - Berechnungsfehler
  - Fehlende oder falsche Programmpfade
- Test auf Robustheit
  - Z.B. Durch Negativtests
- Test der Effizienz
  - Speicherverbrauch
  - Antwortzeiten
- Test auf Wartbartbarkeit (mittels statischer Analyse)
  - Code-Kommentare
  - Numerische Konstanten

3.4.1 Teststrategie - Entwurfskriterien Testbarkeit

- Die Isolierbarkeit einer Komponente ist eine Voraussetzung für Komponententests.
- Isolierbare Komponenten entstehen bei Entwurf nicht zwangsläufig – die Isolierbarkeit muss aktiv im Entwurf herbeigeführt werden.
- Empfehlung: Testbarkeit sollte bei Entwurfsreviews mit einbezogen werden.

3.4.1 Teststrategie – «Test-First»-Ansatz

- Es werden zuerst die Testfälle (in der Regel eingebettet in den Testtreiber) implementiert und anschliessend der produktive Programmcode.
- «Test-First»-Ansatz entstammt den agilen Softwareprozessmodellen (Extreme Programming).
- Vorteile:
  - QS der Anforderung
  - Automatisierung spart Aufwand
  - Der Test verliert den negativen Beigeschmack
  - Testfälle werden dokumentiert und sind reproduzierbar
- In der Praxis stellt eine unvollständige Komponentenspezifikation ein grosses Problem dar!

3.4.2 Integrationstest

- Der Integrationstest bildet die Brücke zwischen dem Komponententest und dem Systemtest.

- **Integration**
  - Der Prozess der Verknüpfung von Komponenten zu grösseren Gruppen.
- **Integrationstest**
  - Testen mit dem Ziel, Fehlerzustände in den Schnittstellen und im Zusammenspiel zwischen integrierten Komponenten aufzudecken.
- **Testbasis**:
  - Softwaredesign, Systemdesign, Systemarchitektur, ggf. auch Workflows oder Use Cases
  -

3.4.2 Testobjekte

- Einzelbausteine zu grösseren Einheiten zusammenbauen
- Alle beteiligten Komponenten, d.h. auch
  - Subsysteme
  - Externe Systeme
  - Datenbanken

3.4.2 Testumgebung

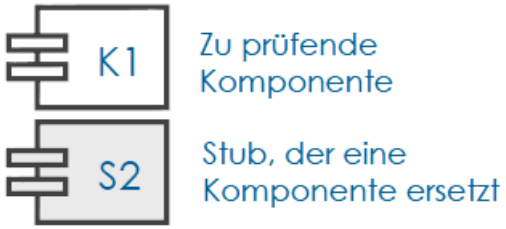
- Analog wie Komponententest
  - Testtreiber
  - Stubs, Mocks etc.
- Zusätzliche Monitore
  - Mitschreiben von Datenbewegungen
  - Standardmonitore für Protokolle

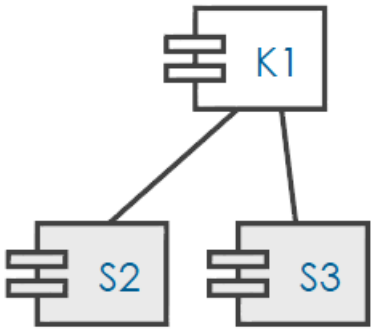
3.4.2 Testziele

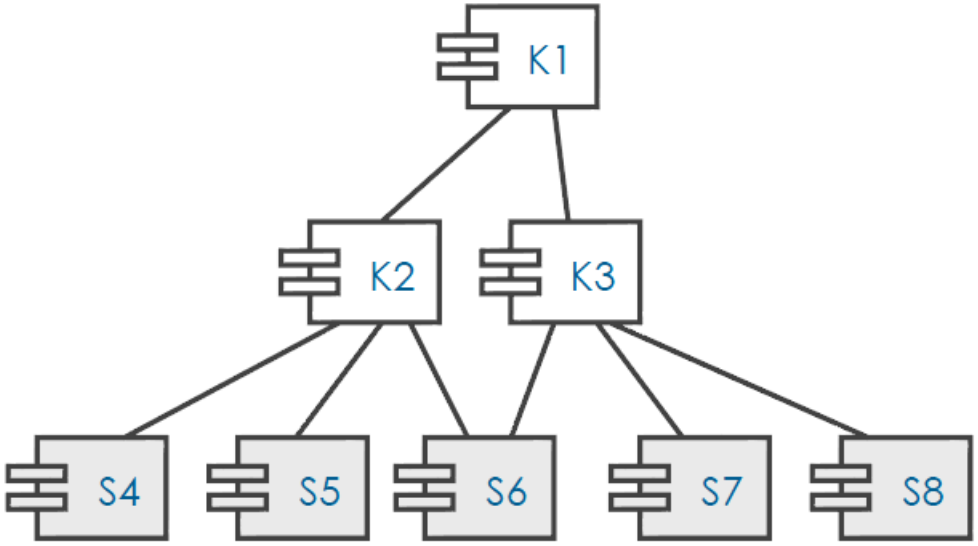
- Ziel des Integrationstests ist es, Schnittstellen- und Protokollfehler aufzudecken.
  - Inkompatible Schnittstellenformate
  - Unterschiedliche Interpretation der übergebenen Daten
  - Timing-Problem: Daten werden richtig übergeben, aber zum falschen oder verspäteten Zeitpunkt oder in zu kurzen Zeitintervallen (Durchsatz- oder Lastproblem)

3.4.2 Integrationsstrategien – Top Down

- Der Test beginnt mit der «Hauptkomponente»
- **Vorteil**: Keine Testtreiber erforderlich
- **Nachteil**: Noch nicht integrierte Komponenten müssen durch Platzhalter ersetzt werden.

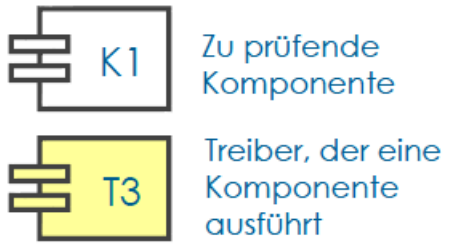


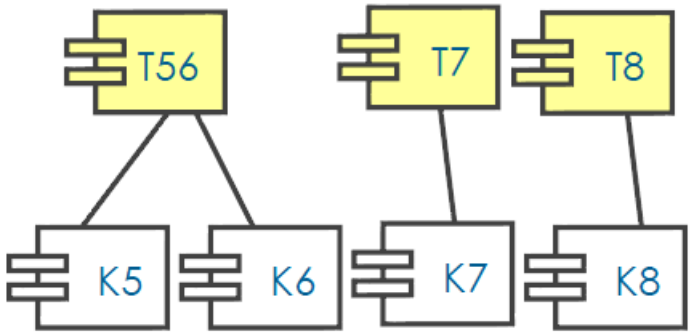


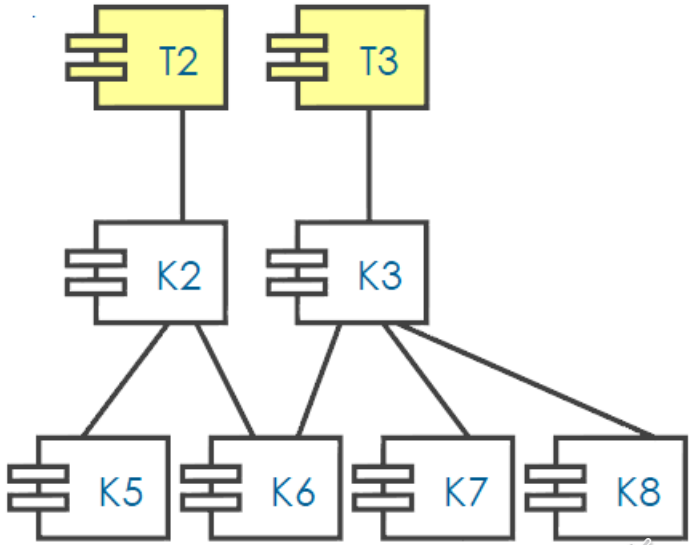


3.4.2 Integrationsstrategien Bottom Up

- Der Test beginnt mit den „untersten“ Komponenten
- **Vorteil:** Keine Platzhalter nötig, „intuitive“ Vorgehensweise
- **Nachteil:** Übergeordnete Komponenten müssen durch Testtreiber simuliert werden. Damit sind viele zusätzliche Testfälle notwendig.

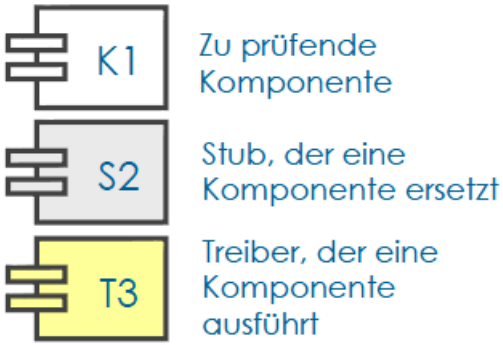


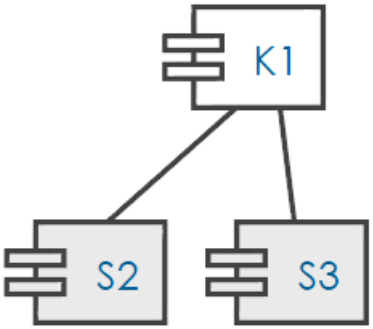


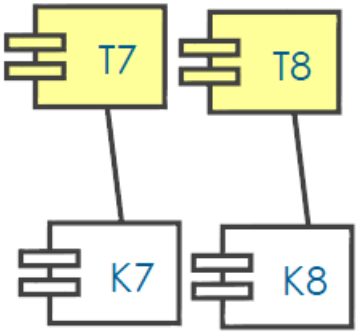


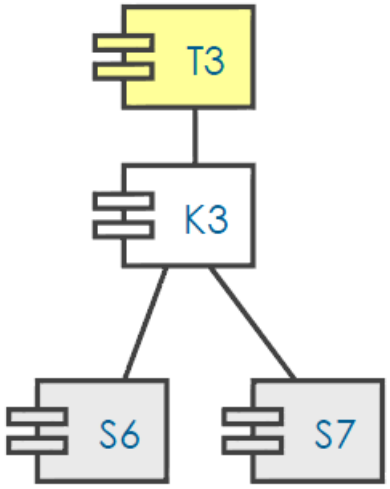
3.4.2 Integrationsstrategien «Ad Hoc»

- Die Komponenten werden z.B. in der (zufälligen) Reihenfolge ihrer Integration fertiggestellt
- **Vorteil:** Fertiggestellte Komponenten können zeitnah integriert werden.
- **Nachteil:** Sowohl Testtreiber als auch Platzhalter sind erforderlich



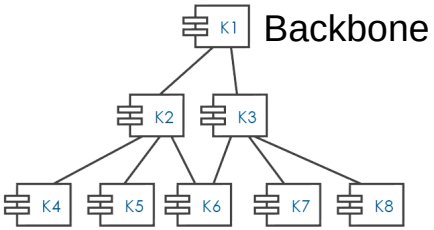






3.4.2 Integrationsstrategien «Backbone-Integration»

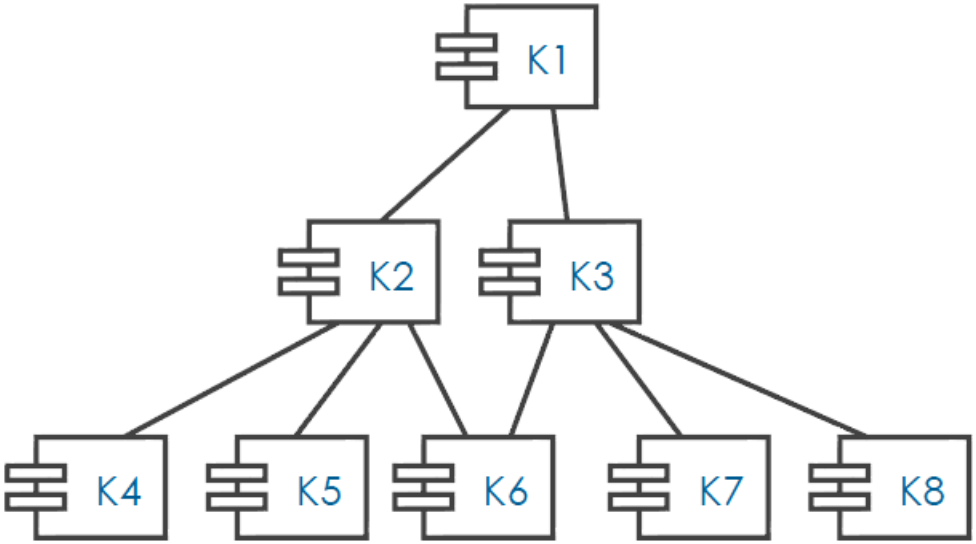
- Es wird ein Programmskelett oder «Backbone» erstellt, in das schrittweise die zu integrierenden Komponenten eingehängt werden.
- «Continuous Integration» (CI) kann als moderne Realisierung dieser Integrationsstrategie gesehen werden, da bei CI ebenfalls ein «Programmskelett» (in Form aller im CI jeweils vorliegenden Systemkomponenten) existiert, das kontinuierlich um die neuen Komponenten erweitert wird.



- **Vorteil:** Komponenten können in beliebiger Reihenfolge integriert werden.
- **Nachteil:** Ein unter Umständen aufwendiger Backbone oder eine CI-Umgebung muss erstellt und gewartet werden.

3.4.2 Integrationsstrategien «Big Bang»

- Der Test beginnt mit dem vollintegrierten System
- **Vorteil:** Es sind keine Testtreiber und Stubs nötig
- **Nachteil:** Schwierige Fehlersuche – alle Fehlerwirkungen treten geballt auf!



3.4.3 Systemtest

- **Test des Gesamtsystems**
  - Test des Zusammenspiels aller integrierten Systemkomponenten
  - Test in produktionsnaher Testumgebung
  - Achtung: Der Begriff „System“ ist keinesfalls eindeutig definiert (SW, SW+HW, SW+HW+Fahrzeug?)
- **Testbasis**
  - Alle Dokumente die das Testobjekt auf Systemebene beschreiben wie Anforderungsdokumente, Spezifikationen, Benutzungshandbücher, etc.

→ *Kundensicht statt Entwicklersicht*

3.4.3 Testobjekt und Testumgebung

- Mit abgeschlossenem Integrationstest liegt das komplett zusammengebaute Softwaresystem vor.
- Systemgrenzen
  - Was gehört alles zum „System“?
  - Die Summe aller Steuergeräte?
  - Ein Steuergerät?
  - Die Software des Steuergeräts?
- Vor Systemtestbeginn (besser vor Projektbeginn) ist diese Frage genauestens zu klären!

3.4.3 Testziele

- Validieren, ob und wie gut das fertige System die gestellten funktionalen und nicht funktionalen Anforderungen erfüllt.
- Fehler und Mängel aufgrund falsch, unvollständig oder im System widersprüchlich umgesetzter Anforderungen sollen aufgedeckt werden.
- Undokumentierte oder vergessene Anforderungen sollen identifiziert werden.
- Prüfung der spezifizierten Systemeigenschaften
  - Test auf Vollständigkeit - auf Basis der Systemspezifikation
- Testen der nichtfunktionalen Anforderungen
  - Leistung bzw. Effizienz
  - Zuverlässigkeit und Robustheit - einschl. Missuse-Testfälle
  - Sicherheit (Security)
  - Benutzbarkeit, Dokumentation

3.4.3 Probleme beim Systemtest

- Der Testaufbau ist aufwändig.
- (blöde, d.h. leicht vermeidbare) Fehler halten den Systemtest immer wieder auf.
- Die Fehlerursache ist aufwändig zu finden.
- Fehler können Schaden an der Betriebsumgebung anrichten (z.B. bei angeschlossenen Geräten).

→ *Um Fehler zu finden ist der Systemtest der ungeeignetste Test! Besser, man findet die Fehler früher.*

3.4.4 Abnahmetest

- Der **Abnahmetest** ist ein spezieller Systemtest
  - Test auf vertragliche Konformität (juristische Relevanz)
  - In der Regel durch den Kunden/Auftraggeber
- Mögliche Resultate
  - Abnahmebescheinigung
  - Nachbesserungsforderungen
  - Rückgabe bzw. Minderung
- Formen des Abnahmetests
  - Test auf vertragliche Konformität
  - Test der Benutzerakzeptanz
  - Akzeptanz durch den Systembetreiber
  - Feldtest (Alpha, Beta, Release Candidate)

3.4.4 Test auf vertragliche Akzeptanz

- Der Kunde führt eine vertragliche Abnahme durch.
- Auf Basis der Ergebnisse entscheidet der Kunde ob das bestellte System den vertraglichen Anforderungen entspricht.
- Testkriterien sind die im Entwicklungsvertrag beschriebenen Abnahmekriterien.
  - Dazu gehören auch Normen und gesetzliche Vorgaben
- Abnahmeumgebung ist normalerweise die Produktivumgebung des Kunden (oder eine Sandbox davon).

3.4.4 Test auf Benutzerakzeptanz

- User Acceptance Test (UAT)
  - Akzeptanz jeder Anwendergruppe sicherstellen
  - Meinungen bzw. Bewertungen der Benutzer einholen
  - Resultate sind oft nicht reproduzierbar
- Mögliches Vorgehen: iterative-inkrementelle Entwicklung – Prototypen frühzeitig den Anwendern vorstellen

3.4.4 Akzeptanz durch den Systembetreiber

- Tests auf Passung in bestehende IT- Infrastruktur
  - Backup, Wiederanlauf
  - Benutzerverwaltung
  - Aspekte der Datensicherheit
- Deployment-Test (Test auf Installierbarkeit)
- Test der Installierbarkeit
  - Die für die Installation zugesicherten Installationsumgebungen werden getestet
  - Virtuelle Umgebungen können den Aufwand zur Bereitstellung der nötigen Testumgebung deutlich reduzieren
- Update-Test
  - Ähnlich dem Deployment-Test
  - Mögliche Ausgangsversion sind zu berücksichtigen
- Test auf Deinstallierbarkeit

3.4.4 Feldtest

- Bei Standardsoftware werden stabile Vorversionen an einen ausgewählten Benutzerkreis ausgeliefert.
  - Alphatests (oft nur beim Hersteller)
  - Betatests
  - Release Candidate
- Benutzer-Feedback muss organisiert sein.
  - User to Supplier (Bugs, new Requirements)
  - Supplier to User

3.5 Grundlegende Testarten

- Folgende grundlegenden Testarten lassen sich unterscheiden:
  - **Funktionale Tests und nicht funktionale Tests**
  - **Anforderungs- und strukturbasierter Tests**
- Fokus und Ziele des Testens variieren von Stufe zu Stufe.
- Dementsprechend kommen verschiedene Testarten in unterschiedlicher Intensität zur Anwendung.

3.5.1 Funktionale Tests

- Funktionalität beschreibt «was» das System leisten soll (Soll-verhalten).
- Funktionalität wird vom System, von einem Teilsystem oder einer Komponente geliefert.
- Testbasis oder Referenz sind: Anforderungsspezifikation, Use Cases, User Stories oder funktionale Spezifikation.
  - Funktionalität kann auch nicht dokumentiert sein (z.B. berechtigte (unterbewusste) Erwartung des Kunden).
- Funktionaler Test prüft das von aussen sichtbare Verhalten der Software (Blackbox-Verfahren, s. Kap. 5.1).
- Verwendung von spezifikationsbasierten Testentwurfsverfahren, um Testendekriterien und Testfälle aus der Funktionalität der Software oder des Systems herzuleiten.

3.5.2 Nicht funktionale Tests

- Nicht funktionaler Test prüft anhand von Software- und Systemmerkmalen, «wie gut» das System arbeitet.
- Nicht funktionale Tests sind z.B.:
  - Lasttest
  - Performanztest
  - Volumen-/Massentest
  - Stresstest
  - (Daten-)Sicherheitstest
  - Zuverlässigkeitstest
  - Robustheitstest
  - Benutzbarkeitstest
  - Interoperabilitätstest
  - Wartbarkeitstest
  - Portabilitätstest
- Grundlage sind Qualitätsmodelle (z.B. ISO 25010).

3.5.3 Anforderungsbezogener und strukturbezogener Test

- Anforderungsbezogenes Testen (anforderungsbasiertes Testen, spezifikationsbasiertes Testen, Blackbox-Verfahren) nutzt als Testbasis Spezifikationen des extern beobachtbaren Verhaltens der Software.
- Anforderungsbezogene Tests werden vorwiegend im System- und Abnahmetest eingesetzt.
- Strukturbezogenes Testen (struktureller Test, strukturbasiertes Testen, Whitebox-Verfahren) nutzt als Testbasis zusätzlich die interne Struktur bzw. Architektur der Software.
- Strukturelle Tests werden vorwiegend im Komponenten- und Integrationstest eingesetzt, in höheren Teststufen manchmal als Ergänzung (z.B. zur Überdeckung von Menüstrukturen).

3.6 Testen nach Änderung und Weiterentwicklung

- Jedes Softwaresystem bedarf über die Dauer seiner Nutzung gewisser Korrekturen und Ergänzungen.
- In diesem Zusammenhang wird von Softwarewartung und Softwarepflege gesprochen.
- Im Gegensatz zu Hardwareprodukten geht es hier jedoch nicht darum, durch regelmässige Pflege die Einsatzfähigkeit zu erhalten oder Schäden, die z.B. durch Abnutzung entstehen, zu reparieren. Denn Software altert und verschleisst nicht.
- Auslöser für die Änderungen eines Softwareprodukts sind die Korrektur von Fehlerzuständen (Hotfixes, Bugfixes) sein oder die geplante Änderung oder Ergänzung von Funktionen im Zuge der «normalen» Pflege oder Weiterentwicklung des Produkts.

3.6.1 Testen nach Softwarewartung und -pflege

- Wurde im Test eine Fehlerwirkung nachgewiesen und korrigiert, muss erneut getestet werden, ob der Fehlerzustand erfolgreich beseitigt wurde (→Fehlernachtest).
- Werden bereits getestete Programmteile einer Änderung unterzogen, ist nachzuweisen, dass durch die Änderung keine Fehlerzustände bzw. Defekte neu eingebaut oder (bisher maskierte) Fehlerzustände dadurch zur Wirkung kommen (→Regressionstest).
- Regressionstests sind auch durchzuführen, wenn sich die Softwareumgebung ändert.
- Nach- und Regressionstests werden oft mehrfach ausgeführt und müssen wiederholbar sein (Kandidaten für Testautomatisierung).
- Regressionstests umfassen funktionale, nicht funktionale wie auch strukturelle Tests (auf allen Teststufen).

Wrap-up

- Das allgemeine V-Modell definiert vier grundlegende Teststufen, Komponententest, Integrationstest, Systemtest, Abnahmetest und unterscheidet zwischen verifizierender und validierender Prüfung.
- Je früher ein Fehler nach seiner Entstehung gefunden wird, desto kostengünstiger ist seine Behebung.
- Agile Softwareprozessmodelle schlagen vor, möglichst auf den unteren Teststufen (Testpyramide) eine hohe, automatisierte Testabdeckung.
- Tests können nach Testzielen unterschieden werden: funktionaler Test, nicht funktionaler Test, strukturbezogener Test und änderungsbezogener Test.
- Durch Wartung, Weiterentwicklung wird ein Softwareprodukt im Laufe seines Lebenszyklus immer wieder geändert oder erweitert und muss darum erneut getestet werden.

Ausblick

- Das Thema der nächsten Vorlesung ist:
  - Kap. 4 Statischer Test (Reviews, statische Analyse)