Advanced Software Engineering 2 - FS21

Pascal Brunner - brunnpa7

Inhaltsverzeichnis

Kapitel 1

Grundlagen des Software Testens

1.1 Begriffe und Motivation

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

Fehlerwirkung / Fehlfunktion / äussere Fehler / Ausfall (engl. failure): Ein Ereignis in welchem eine Komponente od

Fehlerzustand (engl. fault, defect, bug): *Defekt (innerer Fehlerzustand) in einer Komponente oder einem System, der eiene 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)*

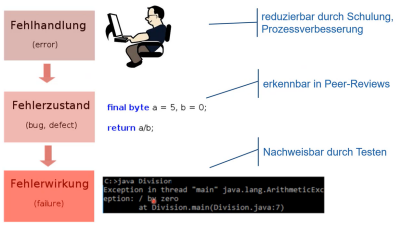

Abbildung 1.1: Abbildung der Fehlerbegriffe

Fehlermaskierung (engl. defect masking): *Ein Umstand, bei dem ein Fehlerzustand die Aufdekcung eines anderen verhindert (nach IEEE 610)*

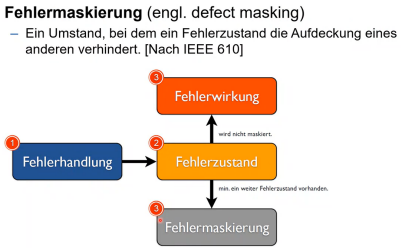

Abbildung 1.2: Abbildung der Fehlermaskierung

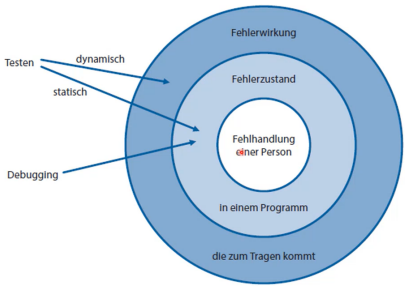

Abbildung 1.3: Zusammenhang der Fehler

False-positive Result: Testergebnis zeigt Fehlerwirkung, obwohl der Fehlerzustand bzw. die Ursache für die Fehlerwirkung nicht im Testobjekt liegt

false-negative Result: Testergebnis zeigt keine Fehlerwirkung, obwohl die Tests diese hätten aufdecken sollen
→ Bei jeder Auswertung von Testergebnissen ist somit zu beachten, ob eine der beiden Möglichkeiten vorliegt

true-positive result: Fehlerwirkung durch den Testfall aufgedeckt

true-negative Result: erwartetes Verhalten bzw. Ergebnis des Testobjekts mit dem Testfall nachgewiesen

1.1.1 Testbegriffe

Um den Defekt zu korrigieren muss der Defekt lokalisiert werden. Bekannt ist die Wirkung aber nicht die genaue Stelle.
Fehlerbereinigung, Fehlerkorrektur (engl. Debugging): Debugging und Testen sind vers. Dinge:

- **Dynamische Tests** können Fehlerwirkung zeigen, die durch Fehlerzustände verursacht werden
- **Debugging** ist eine Entwicklungsaktivtität, die die Ursache (den Fehlerzustand) einer Fehlerwirkung identifiziert, analysiert und entfernt

Ziele des Testens

- Qualititative Bewertung von Arbeitsergebnisse wie Anforderungsspezifikation, User Stories, Design und Programmtext
- Nachweiss, dass alle spez. Anforderung vollständig umgesetzt sind
- Vertrauen in die Qualität
- Höhe des Risikos bei mangelnder Qualität der Software kann durch Aufdeckung von Fehlerwirkungen verringert werden

Unsystematischer Test

- Laufversuch: der Entwickler testet

  Entwickler übersetzt, bindet und startet ein Programm

  Läuft das Programm nicht oder sind Ergebnisse offensichtlich falsch wird Debugging betrieben

  Der Test ist beendet wenn das Programm läuft und Ergebnisse vernünftig aussehen

- Wegwerf-Test: Testen ohne Systematik

  Jemand probiert das Programm mit vers. Eingabedaten aus

  Fallen Ungereimtheiten auf, wird eine Notiz gemacht

  Der Test endet, wenn der Tester findet, es sei genug getestet

1.1.2 Systematischer Test

- Test ist geplant
- Programm wird gemäss Testvorschrifft ausgeführt
- Ist-Resultat wird mit Soll-Test überprüft
- Testergebnisse werden dokumentiert
- Fehlersuche und -behebung erfolgen separat
- Nicht bestandene Test werden wiederholt
- Test endet, wenn vorher definierte Testziele erreicht sind
- Die Testspezfiikation wird laufend aktualisiert

Ziele des systematischen Testens

- Reproduzierbarkeit
- Planbarkeit
- Wirtschaftlichkeit
- Risiko- und Haftungsreduktion

1.2 Testartefakten

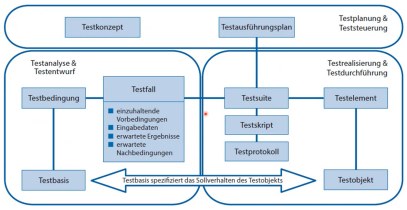

Abbildung 1.4: Uebersicht der Testartefakten und ihre Beziehungen

1.3 Aufwand

- Programm vollständig zu testen ist in der Praxis nicht möglich
- 25-50 Prozent des Entwicklungsaufwands
- Testintensität und umfang in Abhängigkeit mit dem Risiko und Kritikalität
- 2/3 des Testaufwands können auf Komponententests entfallen
- Immer mit beschränkter Ressourcens

1.4 Grundsätze

- Möglichst frühzeitig alle Beteiligten beiziehen
- Beteiligen sich Tester an der Prüfung der Anforderung können Unklarheiten und Fehler in der Arbeitsprodukten aufgedeckt und behoben werden
- Die enge Zusammenarbeit von Testern mit Systemdesiger kann das Verständnis für jede Partei für das Design und des Tests erheblich verbessern
- Arbeiten Entwickler und Tester während der Codeerstellungs zusammen, kann das Verständnis beidseitig verbessert werden
- Wenn Tester die Software vor deren Freigabe verifizieren und validieren können weitere Fehlerzustände erkannt und behoben werden

1. Testen zeigt die Anwesenheit von Fehlerzuständen
2. Vollständies Testen ist nicht möglich
3. Frühes Testen spart Geld und Zeit
4. Häufung von Fehlerzustände → Taucht in einem Modul ein Fehler auf, gibt es eine hohe Wsk dass noch weitere Fehler sich befinden
5. Vorischt vor dem Pestizid-Paradox → nur wiederholen bringen keinen Mehrwert, Testfälle sind zu prüfen, zu aktualisieren und zu modifizieren
6. Testen ist kontextabhängig
7. Trugschluss: Keine Fehler bedeutet ein brauchbares System

Kapitel 2

Testprozess

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 geeignet zu gestalten (Tailoring für ein Projekt)

2.1 Testplanung

- Umfangreiche Aufgabe sollte so früh wie möglich begonnen werden
- Aufgaben und Zielsetzung der tests müssen festelegt werden, genau wie die Ressourcen
- Entsprechende Festlegungen im Testkonzept
  - Testziele
  - Teststrategie
  - Testaktvitäten
  - Ressourcen
  - Testbedingungen und Testbasis
  - Metriken
  - Risiken
- Teststrategie bildet der rote Faden

2.1.1 Testüberwachung und Teststeuerung

- Die Testüberwachung und Teststeuerung umfasst

  fortwährende Beobachtung der aktuell durchgeführten Testaktvitäten im Vergleich zur Planung

  Berichterstattung der ermittelten Abweichungen und die Durchführung der notwendigen Aktivitäten um die Ziele zu erreichen

- Basis für Testüberwachung und -steuerung sind Endekriterien für jeweilige Testaktiväten und -aufgaben

2.1.2 Testanalyse

Bei der Testanalyse geht es darum, zu ermitterln, was genau zu testen ist

- Testbasis prüfen
- Dokumente analysieren
- Testobjekt selbst prüfen
- Berichte heranziehen
- Grundlage ist die Testbasis
- Priorisierung der Testbedingungen
- bidirektionale Rückverfolgung sollte sichergestellt werden (traceability)

2.1.3 Testentwurf

Bei Testentwurf geht es darum, festzulegen, wie getestet wird

- Spezifikation von abstrakten und konkreten Testfällen
- Identifizierung benötigter Testdaten
- Ausgangssituation (Vorbedingung)
- Randbedingungen
- Ergebnisse bzw. welches Verhalten erwartet wird
- Sollergebnis

2.1.4 Testrealisierung

Abschliessende Vorbereitung aller notwendigen Aktivitäten

- Erstellung der Testmittel
- Testrahmen programmiert und Testumgebung installiert
- Abstrtakte Testfälle sind zu konkretisieren
- Testfälle sind zweckmässigerweise zu Testsuiten gruppiert
- automatisierte Testskripts

2.1.5 Testdurchführung

Umfasst die konkrete Ausführung der Tests und deren Protokollierung

- Ausführung von Testabläufen unter Einhaltung des Testplans
- Nachvollziehbarkeit und Reproduzierbarkeit
- Vergleich Ist-Soll
- Fehlerwirkungen oder Abweichungen festhalten

2.1.6 Testabschluss

- letzte Aktivität im Testprozess
- Für Ermittlung von Metriken sollen Testwerkzeuge eingesetzt werden
- unterschiedliche Zeitpunkte für Testabschluss
- Testabschlussbericht ist zu erstellen
  Fasst alle Testaktvitäten und -ergebnisse zusammen
  wird allen Stakeholdern zur Verfügung gestellt
  Testmittel sind zu archivieren

2.2 Psychologie des Testens

- Irren ist menschlich
- Soll der Entwickler sein eigenes Programm testen?
  Blindheit gegenüber eigenen Fehler
  hat hingegen keine Einarbeitungszeit
- Drittperson / Tester
  ist unvoreingenommen
  Einarbeitung notwendig
  Test-Know-how notwendig, bringt ein Tester jedoch mit
- Stufen der Abhängigkeit (von niedrig nach hoch)
  Entwickler selbst
  Kollegen des Entwicklers
  Personen anderer Abteilung
  Person anderer Organisation
- Aufteilung ist produkt- bzw. projektabhängig
- richtige Mischugn und Ausgewogenheit zwischen unabhängiger Tests und Entwicklertests
- Fehlerwirkungen mitteilen
- Reproduzierbarkeit ist wichtig
- Eindeutige Anforderungen, präzise Spezifikation
- Förderlich für die Zusammenarbeit zw. Tester und Entwickler ist die gegenseitige Kenntnis der Aufgaben

Kapitel 3

Testen im Softwareentwicklungslebenszyklus

3.1 Sequentielle Entwicklungsmodelle

Eines der bekanntestens Modellen ist das Wasserfallmodell

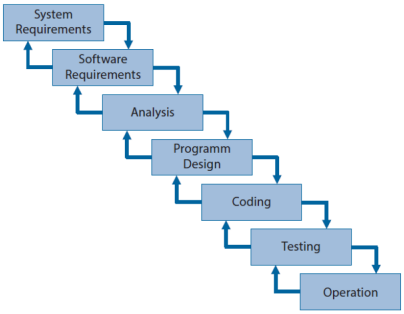

Abbildung 3.1: Das Wasserfallmodell

Ebenfalls zu den sequentiellen Entwicklungsmodellen, gehört das V-Modell

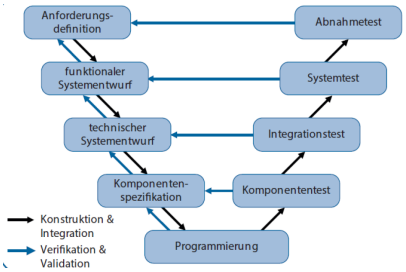

Abbildung 3.2: Das V-Modell

Beim V-Modell trennt man den Konstruktions, mit den Testaktivitäten, sind jedoch gleichwertig aufzufassen (linke und rechte Seite). Dahingehend spricht man von arbeitsteiligen Teststufen, wobei jede Stufe 'gegen' ihre korrespondierende Entwicklungsstufe testet.

Dabei gibt es ein Sprachgebraucht nach ISTQB für das V-Modell

- **Verifikation:** Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind ('Are we doing the thing right?')

  Alternative: Verfikation = formaler Korrektheitsbeweis

- **Validierung:** Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spez. beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind ('Are we doing the right thing?')

Alternative: Validierung = informelle Überprüfung

→ In der Praxis beinhaltet jeder Test beide Aspekte, wobei der validierende Teil mit steigender Teststufe zunimmt

Spricht man von testen innerhalb des Softwareentwicklungslebenszyklus:

- Analyse und Entwurf der Tests während der Entwicklungsaktivtitäten
- Tester sollen im Review Prozess für Requirements oder Architektur als Stakeholder miteingebunden werden

3.2 Iterative und inkrementelle Entwicklungsmodelle

Die wohl bekannteste Form der iterativen Entwicklungsmodellen ist SCRUM.

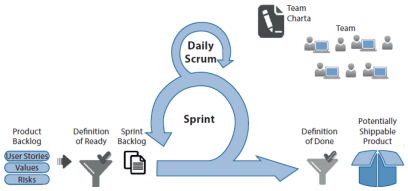

Abbildung 3.3: Scrum

Dabei gibt es die **Testpyramide** nach Cohn und ist eine Hilfe für das Scrum-Team für die Überprüfung, ob die vorhandenen Testfälle angemessen und über sämtliche Teststufen verteilt sind.

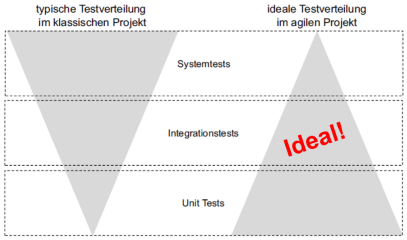

Abbildung 3.4: Die Testpyramide nach Cohn

3.2.1 Testmanagement in Scrum

- Ein Scrum-Team ist ein sich selbst organisierendes, interdisziplinäres Team
- Es gibt keinen Projektleiter, sondern vertraut darauf, dass ein Team sich selbst steuert
- Team ist gemeinsam für alle Arbeiten zuständig
- Programmierung und Testen ist nicht getrennt
- Testmgmt-Aufgaben existieren natürlich
- organisatorische Aufgaben fallen zum Scrum Master
- Leitungsaufgaben im Rahmen der Sprint-Planungspraktiken abgedeckt
- Testaufgaben entweder explizit über eigene Tasks oder implizit mit der Done-Kriterien anderer Aufgaben überwacht
- Testfortschritt und Testergebniss durch Continuous Integration ist hochautomatisiert
- CI kann durch Continuous Delivery erweitert werden → Wenn Tests fehlerfrei durchlaufen, wird es deployed
- klassische Testmanager wird überflüssig, jedoch nicht bei testfachlichen Aufgabenbereichen (Teststrategie und inhaltliches Planen)

- Gemäss Scrum würden diese Aufgaben allensamt mit dem Team durchgeführt
- Ein Teammitglied mit Testexpertise wird benötigt
- Mind. eine Person 'hauptamtlich' für das Testen zuständig sein
- Diese Person berät dann auch der PO bzgl. Produktqualität und Produktfreigabe
- Spricht nichts dagegen diese Person auch Testmanager im Scrum Team zu nennen
- auch exertne Testspezialisten sind öglich
- Auch in Scrum sind grundlegende klassische Testtechniken unverzichtbar
- Alle sollten geschult werden
- Scrum Master oder Testmanager müssten dafür sogar, dass diese Techniken umgeseztt werden

3.3 Softwareentwicklung im Projekt- und Produktkontext

Die Anforderungen an Planung und Nachvollziehbarkeit von Entwicklung undTest sind in unterschiedlichen Kontexten verschieden. Mögliche Punkte können eine Rolle spielen:

- Geschäftsprioritäten
- Art des Produktes
- Markt- und technisches Umfeld
- Identifizierte Produktrisiken
- Organisatorische und kulturelle Aspekte

→ Je nach Einsatzgebiet, sollen die Vorgehensmodelle angepasst werden (Tailoring)

3.4 Teststufen

Beim Testen kann und muss das zu testende System, seine Eigenschaften und sein Verhalten auch auf den versch. Ebenen der Architektur, von den elemanteren Einzelkomponenten bis zum Gesamtsystem, betrachtet und geprüft werden.

Die Testaktivität einer solchen Ebene werden dabei als **Teststufe** bezeichnet. Jede Teststufe ist eine **Instanz des Testprozesses**. Nachfolgend werden die einzelnen Stufen im Detail betrachtet.

3.4.1 Komponententest

Komponenten werden isoliert getestet → Testen im Kleinen

- Das Testobjekt beim Komponententest ist eine Komponente (Unit, Modul, Subsystem)

  Komponente sind ein Teil einer Applikation, bspw. eine Klasse oder Prozedur

  Keine Vorgabe über die Grösse einer Komponente

- Im Komponententest wird die Komponente an den Schnittstellen gegen die Spezifikation und das Softwaredesign getestet.

  es ist eine Komponente-Spezifikation erforderlich!

- typische Testobjekte
  - Komponenten, Klassen(-verbund)
  - Programme
  - Datenumwandlung / Migrationsprogramme
  - Datenbankmodule
- Die Komponente sollte möglichst istoliert getestet werden
- Die Schnittstelle einer Komponente ist i.d.R. eine Programmierschnittstelle
- Wird ein Defekt gefunden, wird das i.d.R. der Komponente zugeordnet.

Testziele

- Test der Funktionalität
  - Berechnungsfehler
- Test auf Robustheit
  - Durch Negativtests
- Test der Effizienz
  - Speicherverbrauch
  - Antwortszeiten
- Test auf Wartbarkeit (mittels statischer Analyse)
  - Code-Kommentare
  - Numerische Konstante

Testbasis:

- Anforderung an die Komponente
- detaillierter Softwaredesign
- Code

Teststrategie

Entwurfskriterien Testbarkeit

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

Test-First Ansatz

- Zuerst alle Testfälle implementieren und anschliessend produktiver Programmcode
- Stammt aus Extreme Programming (agile Methode)
- Vorteile
  - QS der Anforderung
  - Automatisierung spart Aufwand
  - 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

Bildet die Brücke zwischen den Komponenten und dem Systemtest. Wobei als Integration die Verknüpfung der Komponenten zu grösseren Gruppen verstanden wird.

Der dazugehörige Integrationstest hat das Ziel, Fehlerzustände in den Schnittstellen und im Zusammennspiel zwischen integrierten Komponenten aufzudecken

Testbasis

- Softwaredesign
- Systemdesign
- Systemarchitektur
- evtl. Workflows oder Use Cases

Testobjekt

- Einzelbausteine zu grösseren Einheiten zusammenbauen
- Alle beteiligten Komponenten (inkl. Subsysteme, Externe Systeme und Datenbanken)

Testumgebung

- Analog wie Komponententest
- Zusätzliche Monitore
  Mitschreiben von Datenbewegungen
  Standardmonitore für Protokolle

Testziele

- Schnittstellen- und Protokollfehler aufdecken
  inkompatible Schnittstellenformate
  Untersch. Interpretation der übergegeben Daten
  Timing-Problem: Daten werden richtig übergeben aber zum falschen Zeitpunkt

Integrationsstrategien

Top Down

- Der Test beginnt mit der Hauptkomponente
- Vorteil: Keine Testtreiber notwendig
- Nachteil: Noch nicht integrierte Komponenten müssen durch Platzhalter ersetzt werden

Bottom Up

- Der Test beginnt mit der untersten Komponente
- Vorteil: Keine Platzhalter notwendig
- Nachteil: Übergeordnete Komponenten müssen durch Testtreiber simuliert werden

Ad-Hoc

- Die Komponenten werden bspw. in der Reihenfolge ihrer Integration fertiggestellt
- Vorteil: Fertiggestellte Komponenten können zeitnah integriert werden

- Nachteil: Sowohl Testtreiber als auch Platzhalter sind erforderlich

Backbone-Integration

- Es wird eine Programmskette (Backbone) erstellt, in das schrittweise die zu integrierende Komponente eingehängt werden

  Continuous Integration (CI) kann als moderne Realisierung dieser Integrationsstrategien gesehen werden

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

Big Bang

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

3.4.3 Systemtest

Test des Gesamtsytems → Kundensicht statt Entwicklersicht!

- Test des Zusammenspiels aller integrierten Systemkomponenten
- Test in produktionsnaher Testumgebung
- Achtung: der Begriff System ist keinenfalls eindeutig definiertt

Testbasis

- Alle DOkumenten die das Testobjekt auf Systemebene beschreiben wie Anforderungsdokumente, Spezifikation, Benutzungshandbücher etc.

Testobjekt und Testumgebung

- Mit abgeschlossenem Integrationsstest liegt das komplett zusammengebaute Softwaresystem vor
- Systemgrenzen definieren

  Vor Systemtestbgeinng (besser vor Projektbeginn) ist diese Fragen genaustens zu kläre

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 aufdecken
- Undokumentierte oder vergessenee Anforderungen identifizieren
- Prüfung der spezifizierten Systemeigenschaften
- Testen der nicht-funktionalen Anforderungen

Probleme

- Testaufbau ist aufwändig
- (leicht vermeidbare) Fehler halten den Systemtest immer wieder auf
- Die Fehlerursache ist aufwändig zu finden
- Fehler können Schaden an der Betriebsumgebung anrichten (bspw. bei angeschlossenen Geräte)

⇒ Um Fehler zu finden ist der Systemtest der ungeeigeste Test. Man sollte die Fehler früher finden.

3.4.4 Abnahmetest

Spezieller Systemtest → Abnahme des Gesamt- oder Teilsystems durch Auftraggeber

Mögliche Resultate

- Abnahmebescheinigung
- Nachbesserungsforderungen
- Rückgabe bzw. Minderung

mögliche Formen

- Test auf vertragliche Konformität
- test der Benutzeraktzeptanz
- Akzeptanz durch den Systembetreiber
- Feldtest

vertragliche Prüfung

- Der Kunde führt eine vertragliche Abnahme durch
- Basis der Ergebnisse entscheidet der Kunde ob das bestellte System den vertrg. Anforderungen entspricht
- Testkritieren sind im Entwicklungsvertrag beschriebenen Abnahmekriterien (können auch Normen oder gesetzliche Vorgaben sein)
- Abnahmeumgebung ist normalerweise die produktivumgebung des Kunden

Test auf Benutzerakzeptanz

- User Acceptance Test (UAT)

  Akzeptanz jeder Anwendergruppen sicherzustellen

  Meinungen bzw. Bewertungen der Benutzer einholen

  Resultate sind oft nicht reproduzierbar
- mögliches Vorgehen: iterative-inkrementelle Entwicklung - Prototypen frühzeitig den Anwendern vorstellen

Akzeptanz durch den Systembetreiber

- Tests auf Passung in bestehenden IT-Infrastrukturen

  Backup, Wiederanlauf

  Benutzerverwaltung

  Aspekte der Datensicherheit
- Deployment-Test
- Test der Installierbarkeit

  Die für dieInstallation zugesicherten Installationsumgebung werden getestet

  virtuelle Umbgebung können den Aufwand zur Bereitstellung der nötigen Testumgebung deutlich reduzieren
- Update-Test

  Ähnlich Deployment Test

  Mögliche Ausgangsversionen sind zu berücksichtigen

- Test auf Deinstallierbarkeit

Feldtest

- Bei Standardsoftware werden stabile Vorversion an einen ausgewählten Benutzerkreis ausgeliefert

  Alphatest

  Betatests

  Release Candidate

- Benutzer-Feedback muss organisiert sein

  User to Supplier

  Suppplier to User

3.5 Testarten

Folgende grundlegende Testarten lassen sich unterscheiden

- Funktionale Tests und nicht-funktionale Tests
- Anforderungs- und strukturbasierte Tests

Wobei sich der Fokus und die Ziele je nach Stufe variieren. Dementsprechend kommen untersch. testarten in unterschd. Intesität zur Anwendung

3.5.1 funktionale Tests

- Funktionalität beschreibt 'was' das System leisten soll (SOLL-Zustand)
- Funktionalität wird vom System, von einem Teilsystem oder einer Komponente geliefert
- Testbasis oder Referenz sind: Anforderungsspezifikation, Use Case, User Stories oder auch funktionale Spezfiikation
- funktionaler Test prüft das von aussen sichtbare Verhalten der Software (vgl. Blackbox Verfahren)
- Verwendung von spezifikationsbasierten Testentwurfsverfahren, um Testendekriterien und Testfälle aus der Funktionalität der SOftware /System herzuleiten

3.5.2 nicht funktionale Tests

- nicht-funktionale Test prüft anhand von Software- und Systemmerkmalen 'wie gut' das System arbeitet
- Grundlagen gemäss Qualitätsmodelle (bspw. ISO 25010)

  Lasttest

  Performancetest

  etc.

3.5.3 Anforderungsbezogener und strukturbezogener Test

- Anforderungsbezogenes Testen nutzt als Testbasis Spezfiikationen des extern beobachtbaren Verhaltens der Software

  Vorwiegend im System- und Abnahmetest eingesetzt

- strukturbezogenes Testen nutzt als Testbasis zusätzlich die interne Struktur bzw. Architektur der Software

  Vorwiegend im Komponenten- und INtegrationstest eingesetzt

  manchmal in höheren Stufen als Ergänzung

3.6 Test 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
- Auslöser für die Änderungen eines Sofrwareprodukts sind die Korrektur von Fehlerzuständen oder die geplante Änderungen oder Ergänzungen einer Funktion

3.6.1 Testen nach Softwarewartung und -pflege

- Falls Fehlerwirkung bewiesen, muss dieser beseitigt werden (→ Fehlernachtest)
- Bei bereits getesteten Funktionalitäten muss bewiesen werden, dass kein Fehler vorliegt (→ Regressionstest)

  Sind auch durchzuführen, wenn sich die Softwareumgebung ändert
- Nach- und Regressionstest werden oft mehrfach ausgeführt und müssen wiederholbar sein (Kandidaten für Testautomatisierung)
- Regressionstest umfassen funktionale, nicht-funktionale wie auch strukturelle Tests (auf allen Teststufen)