ISTQB TESTANALYST

Zusammenfassung

Version: 1.0

Donnerstag, 6. Oktober 2016

Autor:
Raphael Grüter

FIRMTEC
SOLUTIONS AG

Inhaltsverzeichnis

Historie

Version	Datum	Bemerkungen
1.0	06.10.2016	Initiale Version

Template Version: 1.0

Disclaimer:

FIRMTEC
SOLUTIONS AG

1. Testprozess

1.1. Einführung

- Testprozess: Testplanung, Überwachung, Steuerung / Testanalyse / Testentwurf / Testrealisierung / Testdurchführung / Bewertung Endekriterien und Bericht / Abschluss der Testaktivitäten
- Schwerpunkt für Testanalyst: Festlegung der richtigen Tests/Testfälle, deren Entwurf und Durchführung
- Einfluss auf Testvorgehensweise: Softwarelebenszyklusmodell und Systemtyp

1.2. Testen im Softwarelebenszyklus

- TA muss Zeitpunkte kennen wann Info für andere Stakeholder von Testseite nötig werden:
    - Anforderungsanalyse und Anforderungsmanagement – Anforderungsreviews
    - Projektmanagement – Eingaben für den Zeitplan
    - Konfigurations- und Änderungsmanagement – Tests zur Verifizierung von Softwareversionen, Versionskontrolle
    - Softwareentwicklung – Antizipieren, welche Software wann geliefert wird
    - Softwarewartung – Fehlermanagement, Fehlerbearbeitungszeit (d.h. Zeit von Fehlerfindung bis Fehlerbehebung)
    - Technischer Support – Genaue Dokumentierung von Lösungsalternativen
    - Erstellen technischer Dokumentation (z.B. DB-Designspezifikationen) – Eingaben für diese Dokumente sowie technisches Review dieser Dokumente
- Testaktivitäten an SW Lebenszyklusmodell anpassen: Sequentiell / Iterativ / Inkrementell / Agil
- Beispiel für Anpassung des fundamentalen Testprozesses beim V-Modell auf Stufe Systemtest:
    - Der Systemtest wird gleichzeitig mit dem Projekt geplant, und die Teststeuerung dauert an,
      bis die Testdurchführung und der Abschluss der Testaktivitäten erfolgt sind
    - Systemtestanalyse und -entwurf finden parallel zum Erstellen von Anforderungsspezifikation, System- und (abstraktem) Architekturentwurf statt, und auf einer niedrigeren Ebene gleichzeitig mit dem Komponentenentwurf
    - Die Bereitstellung der Testumgebung (beispielsweise Testrahmen) kann während des
      funktionalen Systementwurfs beginnen, obwohl der größte Aufwand normalerweise
      gleichzeitig mit der Implementierung und dem Komponententest anfällt. Die Arbeit an der
      Systemtest-Realisierung dauert dagegen oft bis wenige Tage vor Beginn der Testdurchführung
    - Die Testdurchführung beginnt, wenn alle Eingangskriterien für den Systemtest erfüllt sind
      (oder auf sie verzichtet wird), was normalerweise bedeutet, dass mindestens der

Komponententest und oft auch der Integrationstest abgeschlossen sind. Die Testdurchführung dauert an, bis die Endekriterien erfüllt sind
    - Während der gesamten Testdurchführung werden die Endekriterien bewertet und die
      Testergebnisse berichtet, normalerweise mit zunehmender Häufigkeit und Dringlichkeit, je
      näher der Projektendtermin rückt
    - Die Testabschlussaktivitäten folgen, wenn die Endekriterien erfüllt sind und die Testdurchführung als abgeschlossen erklärt wird. Sie können aber manchmal verschoben
      werden, bis auch der Abnahmetest abgeschlossen ist und alle Projektaktivitäten beendet sind
- Iterativ/inkrem0entell: Reihenfolge dieser Aufgaben ev. anders oder Teile davon weglassen
- Iteratives Modell: Für jede Iteration eine reduzierte Version des fundamentalen Testprozesses
    - Testanalyse, -entwurf, -realisierung, -durchführung, Bewertung und Bericht in jeder Iteration
    - Planung & Abschlussberichte → Zu Beginn resp. zu Projektende
- Agile Projekte: Leichtgewichtiger Prozess & deutlich engere Zusammenarbeit → Änderungen besser handhabbar
    - Weniger Testdokumentation dafür schnellere Kommunikation (daily Stand-ups)
- Früheste Involvierung des TA's bei Agilen Projekten (VS. Sequentiell & Iterativ)
    - TA ab Anfangsphase involviert → Zusammenarbeit bei Erstellung der ersten Architektur & SW-Entwurf
    - Reviews müssen nicht formal sein → fortlaufende Durchführung im Zuge der SW-Entwicklung
    - TA während gesamten Projektverlauf involviert
- Durch die intensive Beteiligung → feste Zuordnung der Projektmitglieder in den einzelnen Projekten
- Iterative/inkrementelle Lebenszyklusmodelle: von Agilen Modellen bis ins V-Modell eingebettet-iterative
- In Praxis: viele Mischformen, z.B. Iteratives Modell innerhalb V-Modell
    - TA muss häufig die effektivste Rolle bestimmen und sich für diese Rolle einsetzen, anstatt sich auf ein definiertes, festgelegtes Modell verlassen

1.3. Testplanung, -überwachung und -steuerung

1.3.1. Testplanung

- Zusammenarbeit zwischen TA und TM zu folgenden Punkten:
    - TA soll sicherstellen, dass im Testkonzept neben funktionalen Tests auch nicht-funktionale Tests Platz haben (Bsp.: Benutzbarkeit, Wartbarkeit, Übertragbarkeit, Performanz, Sicherheit, Zuverlässigkeit)
    - Testschätzungen absprechen → Genügend Zeit für Beschaffung & Validierung der Testumgebung
    - Planung des Konfigurationstests → Wenn Konfigurationsvariation nötig ist (untersch. Prozessoren, Betriebssysteme, VMs, Browser, Peripherie) → Testverfahren einplanen für dessen Überdeckung

  - Planung der Dokumentationsprüfung → User erhalten SW UND Doku → Doku muss genau & richtig sein → genügend Zeit einplanen, ev. mit technischen Redakteuren zusammenarbeiten (Screenshots, Videos)
  - Installations-, Backup- und Wiederherstellungsprozeduren → ausreichend testen → kann kritischer sein als SW selbst: Falls keine Installation möglich, kein SW Gebrauch → Schwierige Aufgabe für TA, da die ersten Testaktivitäten an vorkonfiguriertem System erfolgt, ohne festgelegten Installationsprozess
  - Testanpassung an SW-Lebenszyklus → Oft parallele/überlappende Arbeiten nötig → TA muss wissen: Welches Modell? Was wird erwartet während Entwurf, Entwicklung und Implementierung der SW?
    → Genug Zeit für Bestätigungs- und Regressionstests einplanen
  - Ausreichend Zeit für Identifizierung & Analyse von Risiken vorsehen. TA nimmt daran teil (→ ohne Lead)
- Aufgabe TA: Komplexes Beziehungsgeflecht zwischen Testbasis, Testbedingungen und Testfällen bestimmen und Abhängigkeiten so weit wie möglich voneinander trennen

1.3.2. Testüberwachung und -steuerung

- Testüberwachung und -steuerung = Aufgabe vom TM → TA trägt Messungen bei um Steuerung zu ermöglichen
- Sammlung von quantitativen Daten während gesamtem SW-Lebenszyklus, z.B. %-Anteil abgeschlossener Planungsaktivitäten, %-Anteil der erzielten Überdeckung, # TCs bestanden / nicht bestanden
- Definition einer Baseline (Referenzkonfiguration) nötig → Messung des Fortschritts
- TM → Erstellen und Berichten der Metriken, TA → Sammlung der dafür benötigten Informationen
- Projektmetriken → Jeder abgeschlossene Testfall, Fehlerbericht, Meilenstein fliesst da rein
- Genaue Metriken = bessere Überwachung und Steuerung der Aktivitäten, z.B.:
  - SW in einem Bereich viele Fehler → dort mehr Testaufwand nötig
  - Überdeckung von Risiken / Anforderungen → Priorisierung der noch ausstehenden Arbeiten
  - Infos über Grundursachen → Identifikation von Prozessverbesserungen möglich
  - Bessere Planung möglich, wenn Daten aus vorhergehenden Projekten herangezogen werden können

1.4. Testanalyse

- TA nutzt die Testplanung für: Analyse der Testbasis & Identifizierung der Testbedingungen
- Für eine effektive Durchführung der Testanalyse → Erfüllung folgender Eingangskriterien nötig:
  - Beschreibendes Dokument über Testobjekt vorhanden → Dient als Testbasis
  - Review dieses Dokumentes liefert brauchbare Ergebnisse und wird danach ggf. aktualisiert
  - Für Durchführung der restlichen Aktivitäten zu diesem Testobjekt gibt es angemessenes Budget & Zeit
- Identifikation der Testbedingungen durch Analyse der Testbasis

- Falls Doku veraltet / nicht vorhanden → Ident. der Testbed. durch Gespräche (z.B. Workshops / Sprintplanung)
- Dann wird bestimmt, was mit den in der Teststrategie / Testkonzept festgelegten Testverfahren getestet wird
- Testbedingungen i.d.R. für das zu testende Element / Objekt → Grunds. Beachtung folgender Grundsätze:
    - Es empfiehlt sich den Detaillierungsgrad der Testbed. Unterschiedlich zu halten. Zuerst die abstrakten Bedingungen (→ allgem. Ziele definieren, z.B.: «Nachweisen, dass Bildschirm X funktioniert). Danach detailliertere Bedingungen als Basis für spezifische Testfälle identifizieren (z.B.: «Nachweisen, dass Bildschirm X eine Kontonummer zurückweist, die ein Zeichen zu wenig hat.» → Dadurch Sicherstellung, dass Abdeckung der abstrakten Elemente ausreichend ist
    - Falls Produktrisiken vorhanden → Identifikation der für jedes Risiko notwendigen Testbedingungen
      → Rückverfolgbarkeit zu den Risiken muss möglich sein
- Nach Abschluss der Testanalyse → TA weiss, welche spezifischen Tests entworfen werden müssen, damit Erfordernisse der festgelegten Bereiche des Testprojekts abgedeckt werden

1.5. Testentwurf

- Auch Testentwurf basiert auf dem in der Testplanung definierten Umfang
- TA entwirft die Tests, die im weiteren Verlauf des Testprozesses realisiert & durchgeführt werden:
    - Bestimmung von Bereichen wo abstrakte und Bereiche wo konkrete Testfälle geeignet sind
    - Festlegung der Testfallentwurfsverfahren, welche die benötigte Testabdeckung liefern
    - Entwurf der Testfälle, mit denen die identifizierten Testbedingungen ausgeführt werden
- Priorisierungskriterien aus Risikoanalyse & Testplanung → Im gesamten Testprozess angewendet (T-A, -E, -R, -D)
- Eingangskriterien für Testentwurf → z.B. Verfügbarkeit von Werkzeugen, für Entwurfsaufgaben genutzt
- Folgende Punkte sind beim Testentwurf zu berücksichtigen:
    - Für manche Testelemente ist es besser, nur die Testbedingungen zu spezifizieren, anstatt detaillierte Testablaufbeschreibungen zu definieren. In diesen Fällen sollten die Testbedingungen so spezifiziert werden, dass sie als Leitfaden für Tests ohne Testablaufspezifikation verwendet werden können
    - Die Endekriterien für die Tests müssen eindeutig festgelegt werden
    - Tests sollten so entworfen werden, dass sie für andere Tester verständlich sind, und nicht nur
      für den Autor. Falls der Autor nicht die Person ist, die den Test durchführt, dann müssen
      andere Personen die zuvor spezifizierten Tests lesen und verstehen, damit sie die Testziele
      und die relative Wichtigkeit des Tests nachvollziehen können

- Die Tests müssen auch für andere Stakeholder verständlich sein, beispielsweise für Entwickler, da diese die Tests prüfen werden, und für die Leiter eines Audits, die die Tests eventuell genehmigen müssen
- Tests sollten so entworfen werden, dass sämtliche Interaktionen der Software mit Akteuren (z.B. Endnutzer, andere Systeme) abgedeckt sind, und nicht nur die Interaktionen, die über die Benutzerschnittstelle für den Nutzer sichtbar sind. Auch Kommunikationen zwischen Prozessen, Batch-Verarbeitung und Interrupts beeinflussen die Software und können potenziell Fehlerwirkungen auslösen. Daher muss der Test Analyst auch Tests entwerfen, die diese Risiken beherrschen
- Tests sollten entworfen werden, um die Schnittstellen zwischen den verschiedenen Testobjekten zu testen

1.5.1. Logische und konkrete Testfälle

- Logische Testfälle: Richtlinie, spezifizieren WAS getestet werden soll. Tatsächliche Daten & Ablauf ist offen
  - Vorteil: Bessere Abdeckung, da bei jeder Durchführung variiert wird, früher Einsatz möglich
  - Nachteil: Geringere Reproduzierbarkeit
  - Einsatzgebiet: Bei schlecht spezifizierten Anforderungen, Erfahrung des Testers bez. Testing & Testobjekt, keine Formale Doku nötig (→ keine Audits), logischer TC → konkreter TC (früh → später)
- Konkrete Testfälle: Liefern alle spezifischen Informationen & Prozeduren für Testdurchführung & Verifizierung der Ergebnisse, inkl. der benötigten Daten
  - Vorteil: Nützlich wenn Anforderungen gut spezifiziert sind, Tester wenig Erfahrung hat, für Audits, hohe Reproduzierbarkeit
  - Nachteil: Wartungsaufwand kann erheblich höher sein, Einengung der Phantasie des Testers

1.5.2. Testfälle entwerfen

- Testfälle werden in schrittweiser Ausarbeitung & Verfeinerung der identifizierten Testbedingungen entworfen
- Dafür nutzen Tester Testverfahren, die in Teststrategie &/ Testkonzept festgelegt werden
- Testfälle sollen sein: wiederholbar, nachprüfbar, rückverfolgbar (zur Testbasis, z.B. Anforderungen)
- Zum Entwurf der Testfälle gehört die Identifizierung von:
  - Zielsetzung
  - Vorbedingungen: Testumgebungen, -vorrichtungen, Zustand des Systems, etc.
  - Anforderungen für Testdaten: Testfall-Eingabedaten, Daten im System
  - Erwartete Ergebnisse
  - Nachbedingungen: beeinflusste Daten, Systemzustand, Auslöser für folgende Prozessabläufe
- Detaillierungsgrad: Wirkt sich auf die Entwicklungskosten & Wiederholbarkeit aus. Sollte daher vor Testfallerstellung festgelegt werden

- Weniger Details → Mehr Flexibilität für TA bei Durchführung → interessantere Bereiche genauer anschaubar
- Definition der erwarteten Ergebnisse oft besondere Herausforderung → manuelle Berechnung oft schwierig
  → Wenn möglich automatisiertes Testorakel finden / erstellen
- Nicht nur Bildschirmausgaben beachten, sondern auch auf Daten und Nachbedingungen in Testumgebung
- Falls Testbasis klar definiert → Identifikation der Ergebnisse theoretisch einfach, in Realität ist Testbasis jedoch oft vage, widersprüchlich, nicht alle Schlüsselbereiche abdeckend, unvollständig oder gar nicht vorhanden
  → TA braucht/muss Zugang haben zu fachlicher Expertise
- Auch wenn Testbasis klar definiert → Komplexe Interaktionen zwischen Stimuli & Reaktion → erwartetes Testergebnis kann trotzdem schwierig sein → Testorakel unverzichtbar
- Ohne Überprüfbarkeit der Ergebnisse hat der Test nur begrenzten Wert → ungültige Fehlerberichte / falsches Vertrauen in System
- Die beschriebenen Aktivitäten finden auf allen Stufen statt, es ändert nur die Testbasis:
  - Anwender-Abnahmetests: Anforderungsspezifikation, Anwendungsfälle, definierte Geschäftsprozess
  - Komponententests: detaillierte Entwurfsspezifikation, User Stories, Programmcode selbst
- Ziele des Testens:
  - Funktionaler Komponententest: Komponente liefert Funktionalität, spezifiziert im Detailentwurf
  - Funktionaler Integrationstest: Zusammenwirken der Komponenten
  - Systemtest: End-to-End Funktionalität
- Bei Analyse & Entwurf → Stufe und Testobjekt berücksichtigen → hilft den benötigten Detaillierungsgrad & Werkzeuge zu bestimmen (z.B. Treiber & Platzhalter bei Komponententests)
- Detaillierungsgrad von Arbeitsergebnissen wird beeinflusst durch:
  - Projektrisiken: Was muss dokumentiert werden?
  - Mehrwert, den die Doku für das Projekt schafft
  - Standards &/ Vorschriften
  - Lebenszyklusmodell: bei agilen Projekten → gerade ausreichende Doku
  - Anforderung an Rückverfolgbarkeit: Testbasis → Testanalyse → Testentwurf
- Je nach Testumfang: Testanalyse- & Testentwurfsphase befasst sich auch mit Qualitätsmerkmalen des Testobjektes → ISO 25000 (alt: ISO 9126) = nützliche Referenzgrundlage
- Testanalyse- & Testentwurfsprozess: Verbesserung durch Verknüpfung von Reviews & statischer Analyse
- Testanalyse & Testentwurf = statisches Testen der Anforderungen
- Auch Testfälle, Risikoanalyse und Testkonzepte sollten Reviews & statischen Analysen unterzogen werden
- In Testentwurfsphase → Anforderungsdetails an Testinfrastruktur genau definieren, in Praxis werden diese jedoch erst zu Beginn der Testrealisierung endgültig festgelegt.
- Testinfrastruktur: Testobjekte, Testmittel, Räumlichkeiten, Ausrüstungen, Personal, Software, Werkzeuge, Peripheriegeräte, Kommunikationseinrichtungen, Zugriffsberechtigungen, etc.

- Endekriterien von Testanalyse & Testentwurf: variieren je nach Projektparameter, sollten messbar sein, alle Infos und Vorbereitungen für nachfolgende Schritte sollen zur Verfügung gestellt werden

1.6. Testrealisierung

- Testrealisierung = Umsetzung des Testentwurfs: Erstellung automatisierter Tests, Festlegung der Reihenfolge für die Ausführung der manuellen & automatisiertes Tests, endgültige Festlegung der Testdaten und -umgebungen, Erstellung eines Testausführungsplans inkl. Ressourcenzuweisung
- Überprüfung der expliziten & impliziten Eingangskriterien der betroffenen Teststufe
- Sicherstellen, dass die vorangegangenen Prozessschritte erfüllt wurden:
    - Falls nicht: Ev. Verzögerungen, mangelnde Qualität, unerwarteter Mehraufwand → Wichtig, dass alle Endekriterien erfüllt sind, bevor mit Testrealisierung begonnen wird
- Bei Festlegung der Reihenfolge zu beachten: Zusammenstellung der Testfälle in Testsuiten manchmal sinnvoll
    - Testsuite: Gruppe von Testfällen, die zusammengehören, miteinander ausführen
- Bei risikoorientierter Teststrategie → Reihenfolge durch Risikopriorität gegeben
- Noch viele andere Faktoren wichtig für Reihenfolge: Verfügbarkeit der Personen, Ausrüstungen, Daten, zu testende Funktionalität
    - Nicht ungewöhnlich, dass Code schrittweise freigegeben wird & Testaufwand mit Reihenfolge koordiniert werden muss (v.a. bei inkrementellen Lebenszyklusmodellen)
- Bei Testrealisierung soll TA die Reihenfolge der man. & autom. Tests endgültig festlegen & bestätigen
    - Sorgfältig prüfen, ob Faktoren vorhanden, die bestimmte Reihenfolge zwingend vorgeben
    - Abhängigkeiten müssen dokumentiert & überprüft werden
- Detaillierungsgrad/Komplexität bei Testrealisierung = f (Detaillierungsgrad der Testfälle & -bedingungen)
- Testdaten: Nötig fürs Testen, Datenmengen können umfangreich sein, Umgebungs- & Eingabedaten → abgelegt in DB's o.ä., TA erzeugt Daten für datengetriebene Tests (man. & autom.)
- Testrealisierung umfasst auch Testumgebung(en) → sollen vollständig vorhanden & verifiziert sein
    - Fehlerzustände sollen aufgedeckt werden können, sollte normal operieren solange keine Fehlerwirkungen auftreten, sollte bei Bedarf auf höheren Teststufen die Produktions- oder Anwendungsumgebung angemessen abbilden
- Änderungen an Testumgebung nötig während Testdurchführung?
  → Auswirkung auf bereits ausgeführte Tests prüfen
- Personen zuständig für Erzeugung und Wartung der Testumgebung: bekannt & verfügbar
- Während TR sicherstellen: Testmittel & Werkzeuge zur Testunterstützung und zugehörige Prozesse einsatzbereit:
    - Konfigurationsmanagement, Fehlermanagement, Testprotokollierung, Testmanagement
- Während TR: Verfahren verifizieren, mit welchen die Daten für Testendekriterien & Berichte über Testergebnisse gesammelt werden

- Oft ausgewogener Ansatz gewählt (Risikoorientiert & Analytisch, kombiniert mit dynamischen Tests)
- Dynamische Tests:
  - Fehlerangriffe, intuitive Testfallermittlung, exploratives Testen
  - Testanalyse, Testentwurf, Testrealisierung → bei Testdurchführung, nicht vorab
  - Zeitliche Eingrenzung nötig, um Abdriften zu vermeiden
  - Die Testergebnisse beeinflussen TA, TE, TR der nachfolgenden Tests
  - Leichtgewichtige Teststrategie
  - Effektiv beim Fehleraufdecken, Tester braucht viel Erfahrung, Dauer schwer abschätzbar, Infos über Testüberdeckung oft unvollständig, ohne gute Doku / spez. Regressionswerkzeug schwer reproduzierbar

1.7. Testdurchführung

- TD beginnt, sobald das Testobjekt zur Verfügung steht & Eingangskriterien erfüllt sind / darauf verzichtet wird
- TD gemäss Plan aus TR, jedoch mit gewissem Spielraum für Tester, damit zusätzliche interessante Testszenarien & Testverhalten verfolgt werden können, die sich erst während des Testens ergeben
- Werden Fehlerwirkungen bei Abweichung vom festgelegten Testvorgehen aufgedeckt → Doku so schrieben, dass Testvorgehen reproduzierbar ist
- Einbindung von Testverfahren mit/ohne Testablaufbeschreibung (z.B. explorative Verfahren) hilft zu verhindern, dass Bereiche nicht abgedeckt werden, die sich aus Lücken der vorgeschriebenen Überdeckung ergeben. Hilft zudem das Problem zu entschärfen, dass Testwiederholungen zunehmend an Wirksamkeit verlieren
- Kernstück der Testdurchführung = Vergleich tatsächlicher VS. erwartetem Testergebnis
  - Falsch Negativ → Fehler übersehen, Falsch Positiv → irrtümlicher Fehler entdeckt
  - Abweichung, wenn Erwartungswert =! Messwert → Fehler im Testobjekt?
  - Abweichung entdeckt → 1. Überprüfung der Doku (Testspez., Testfall, usw.)
  - Änderung an Testbasis &/ Testobjekt → kann dazu führen, dass Testspezifikation inkorrekt wird
- Protokollierung der Testergebnisse während Testdurchführung nötig
- Falls Test ausgeführt aber nicht protokolliert → ev. Testwiederholung nötig
- Da Testobjekt, Testmittel, Testumgebung ändern können → Version muss aus Testprotokollierung hervorgehen
- Testprotokollierung liefert chronologische Aufzeichnung aller relevanten Details der Testdurchführung
- Testprotokoll liefert Infos für: Testüberdeckung, Gründe für Verzögerungen, Teststeuerung, Testfortschrittberichte, Testendekriterien, Verbesserungen des Testprozesses
- Protokollierung = f (Teststufe, Teststrategie), Vorschriften/Audit-Anforderungen → Vorgaben für Protokollierung
- Nutzer / Kunden beteiligen an Testdurchführung kann Vertrauen stärken, Fehlerfindung muss minimiert sein! Trifft in frühen Teststufen selten zu, während Abnahmetests eher
- Folgende Punkte sind zu berücksichtigen bei Testdurchführung:
  - «Irrelevante» Merkwürdigkeiten (Nebeneffekte) → wahrnehmen & untersuchen
  - Sicherstellen, dass Produkt nichts tut was es nicht tun soll

- Testsuite-Anpassungen = kontinuierlicher Prozess → Code wächst = neue Funktionen = mehr Tests. Lücken in Testabdeckung werden häufig während Testdurchführung entdeckt
- Notizen für kommende Testaufgaben nötig, auch nach SW Release (→ neue SW Releases)
- Wiederholung aller manuellen Tests unwahrscheinlich → Problem vermutet? Untersuchen!
- Testfälle ergänzen/anpassen mit Infos aus Fehlerverfolgungswerkzeug
- Regressionstests: Fehler sollten vorher entdeckt werden, da Zeit für Regression oft beschränkt

1.8. Bewertung von Endekriterien und Bericht

- Für Überwachung des Testfortschritts im Testprozess geht es v.a. darum, Infos gemäss Anforderungen für Berichte zu sammeln, inkl. Testfortschritt in Bezug auf Endekriterien messen
- Endekriterien → was MUSS resp. was SOLL erfüllt werden? Z.B.: Keine Fehlerzustände der Klassen 1 & 2 und mindestens 95% aller Testfälle müssen bestanden werden → wenn dann z.B. nur 93% erfüllt, kann Projekt trotzdem für nächste Stufe freigegeben werden. Endekriterien müssen klar spezifiziert sein, damit objektive Bewertung möglich
- TA liefert Info, die TM zur Bewertung des Testfortschritts gegen Endekriterien verwendet: Daten richtig?
- Falls TM-System z.B. folgende Statis vorgibt: Bestanden / Nicht bestanden / Bestanden mit Ausnahme → TA muss wissen was diese Statis bedeuten, und konsistent anwenden: Bestanden mit Ausnahme = Fehlerzustand gefunden, welcher die Funktionalität nicht beeinträchtigt? Was ist mit Benutzbarkeitsfehlerzuständen, die den Benutzer verwirren? → Für Erfüllungsquote relevant, ob Testfall = Nicht bestanden / Bestanden mit Ausnahme
- Testfall = Nicht bestanden, weil z.B. falsch konfigurierte Testumgebung: wie behandeln? → Klärung TA & TM
- Durchaus üblich, das TA um Statusbericht während Testzyklen und für Abschlussbericht gebeten wird → dafür werden Metriken aus Fehler- und Testmanagementsystem gesammelt, Gesamtüberdeckung und Fortschritt werden bewertet
- TA sollte Berichtswerkzeuge benutzen können & fähig sein, dem TM die angeforderten Infos zu liefern

1.9. Abschluss der Testaktivitäten

- Nach Abschluss Testdurchführung → wichtigste Ergebnisse des Testens festhalten & archivieren oder an zuständige Person weiterleiten = Abschluss der Testaktivitäten → TA darin involviert
- Support &/ Kunden etc. sollen über offene Fehler informiert werden
- Personen für Wartungstests sollen Tests & Testumgebungen erhalten
- Satz von man. & autom. Tests / Regressionstests = weiteres Arbeitsergebnis
- Infos über Testarbeitsergebnisse klar dokumentieren, inkl. Links & Zugriffsberechtigungen
- TA nimmt auch an Lessions Learned teil → Themen: Testprozess, SW-Lebenszyklus
  - Falls nur TM teilnimmt → TA muss Infos an TM zur Verfügung stellen

- Ins Konfigurationsmanagement-System zu archivieren: Ergebnisse, Protokolle, Berichte, andere Doku & Arbeitsergebnisse → Oft TA dafür zuständig
  → Wichtige Testabschlussaktivität, v.a. wenn diese Info für zukünftiges Projekt Verwendung findet
- TM bestimmt i.d.R. welche Info archiviert wird, TA sollte überdenken, welche Info benötigt würden, falls Projekt erneut gestartet wird, ev. sogar mit neuem Testteam → kann Monate an Aufwand ersparen

2. Testmanagement: Zuständigkeiten des Test Analysten

2.1. Einführung

- Es wird erwartet, dass TM die benötigten Infos vom TA anfordert → Hohlschuld vom TM

2.2. Testfortschritt überwachen und steuern

- 5 wesentlichen Aspekte für Überwachung des Testfortschritts: Produktrisiken, Überdeckungsgrad, Tests, Fehlerzustände, Vertrauen in SW
- TA misst & berichtet über Produktrisiken, Überdeckungsgrad, Testfälle, Fehlerzustände. Vertrauen in SW ist oft subjektiv, auch wenn durch Umfragen messbar
- Zusammentragen der Infos, auf welche sich die Metriken stützen = tägliche Aufgabe des TA
- Richtigkeit der gesammelten Daten kritisch → ungenaue Daten = falsche Trends = falsche Schlussfolgerungen
- Bei risikoorientiertem Testen → Metriken, die TA verfolgen soll:
  - Welche Risiken wurden durch Testen reduziert?
  - Welche Risiken gelten als offene Risiken?
- Risikobeherrschung → häufig mit Hilfe eines Werkzeuges, das auch Testdurchführung verfolgt (z.B. TM-Werkzeug)
  - Bedingt, dass die identifizierten Risiken mit Testbedingungen in Bezug gesetzt werden, welche sich wiederum auf die Testfälle beziehen die die Risiken reduzieren, wenn sie erfolgreich ausgeführt werden
  - Dadurch werden die Infos zur Risikoüberwachung automatisch mit Testfallaktualisierung aktualisiert
- Fehlerverfolgung erfolgt i.d.R. durch Fehlermanagementwerkzeug. Neben Fehlerzustände werden Infos betreffend Fehlerklasse jedes einzelnen Fehlerzustandes aufgezeichnet → daraus werden Trenddiagramme und Grafiken erstellt, aus denen der Testfortschritt & SW-Qualität hervorgehen
- Lebenszyklus kann Umfang der aufzuzeichnenden Fehlerdoku & Methoden zur Aufzeichnung der Info beeinflussen
- Bei Testdurchführung sollte Statusinformation zu Testfällen aufgezeichnet werden → i.d.R. durch TM-Werkzeug, bei Bedarf auch manuell. Zu Testfallinformationen können gehören:
  - Status der erstellen Testfälle (z.B. entworfen, geprüft)
  - Status der durchgeführten Testfälle (z.B. (nicht) bestanden, blockiert, ausgelassen)

    - Informationen über durchgeführte Testfälle (z.B. Datum & Uhrzeit, Tester, verwendete Daten)
    - Artefakte der durchgeführten Testfälle (z.B. Screenshots, testbegleitende Protokolle)
- Neben Risiken sollen auch Testfälle mit den jeweiligen Anforderungen in Bezug gebracht werden, die sie abdecken
- TA soll beachten: wenn Testfall A sich als einziger Testfall auf Anforderung A bezieht → Anforderung gilt als erfüllt, sobald Testfall A bestanden wird. In vielen Fällen wird mehr als nur ein Testfall benötigt, um eine Anforderung gründlich zu testen, aber aus Zeitmangel wird nur ein Teil davon tatsächlich erstellt. Bsp.: 20 Testfälle sind nötig, um Implementierung einer Anforderung gründlich zu testen, aber nur 10 davon sind entworfen & ausgeführt → Überdeckung = 100 %, auch wenn tatsächlich nur 50 % erzielt wurde → Genaue Verfolgung der Überdeckung sowie der Anforderungen mit Status «geprüft» → Vertrauensbildende Massnahme
- Umfang & Detaillierungsgrad der aufgezeichneten Info = f (u.a. SW-Lebenszyklus)
- Agile Projekte → weniger Statusinfo aufgezeichnet, weil Team enger zusammenarbeitet & mehr Kommunikation

2.3. Verteiltes Testen, Outsourcing und Insourcing

- Verteiltes Testen = Testaufwand wird an mehreren Standorten erbracht
- Zentralisiertes Testen = Testaufwand wird an einem einzigen Standort erbracht
- Outsourcing = Testaufwand an einem oder mehreren Standorten (jedoch nicht am Standort des Projektteams), durch Tester die nicht Mitarbeiter des Unternehmens sind
- Insourcing = Testaufwand durch Mitarbeiter ausserhalb des Projektteams, aber am selben Standort
- Testaufwand an unterschiedlichen Standorten → besondere Aufmerksamkeit auf effektive Kommunikation
- Bei 24h-Testerei an verschiedenen Standorten → Arbeitsübergabe von einem Team einer Zeitzone an das nächste → spezielle Planung seitens TA's nötig, die die Arbeit übergeben / entgegen nehmen → Gute Planung zwar wichtig um Zuständigkeiten zu verstehen, doch es ist unerlässlich, dass TA sicherstellt, dass richtige Infos verfügbar sind
- Wenn Kommunikation nicht mündlich erfolgen kann, muss schriftlich ausreichen → Statusberichte, E-Mails und effektive Nutzung von TM- und Fehlermanagementwerkzeug
- Wenn im TM-Werkzeug Tests einzelnen Personen zugeteilt werden können → TM-Werkzeug = Planungswerkzeug, bietet einfache Möglichkeit, Arbeit zwischen Personen zu übergeben
- Fehlerzustände, die genau aufgezeichnet werden → bei Bedarf an Kollegen zur Nachverfolgung weitereiten
- Effektive Nutzung dieser Kommunikationsmittel für Organisationen, die keine tägliche persönliche Interaktion zulassen, von entscheidender Bedeutung

2.4. Aufgaben des Test Analysten beim risikoorientierten Testen

2.4.1. Übersicht

- TM = Gesamtverantwortlich für Aufsetzen und Management einer risikoorientierten Teststrategie
- I.d.R. wird TM die Unterstützung des TA anfordern um sicherzustellen, dass risikoorientierte Vorgehensweise fachgerecht implementiert wird
- TA hilft bei folgenden risikoorientierten Testaufgaben: Risikoidentifizierung, -bewertung, -beherrschung
- Diese Aufgaben erfolgen iterativ über ges. Projektlebenszyklus → befasst sich mit auftretenden Risiken, Änderung von Prioritäten, regelmässige Bewertung und Kommunikation des Risikostatus
- Risiken des Geschäftsbereichs= Betriebssicherheit, geschäftliche & wirtschaftliche Aspekte, politische Faktoren

2.4.2. Risikoidentifizierung

- Je breiter die Basis der Involvierten Personen, desto höher die Wahrscheinlichkeit, dass die grösstmögliche Menge an wichtigen Risiken aufgedeckt wird
- TA verfügen häufig über spezifisches Wissen über den jeweiligen Geschäftsbereich → besonders geeignet um:
    - Interviews mit Experten & Benutzern des Geschäftsbereichs zu führen
    - Unabhängige Bewertungen zu erstellen
    - Risikovorlagen zu verwenden & deren Einsatz zu ermöglichen
    - Risikoworkshops zu leiten
    - Brainstorming mit potentiellen & aktuellen Nutzern durchzuführen
    - Checklisten zu spezifizieren
    - Auf Erfahrungen aus der Vergangenheit mit ähnlichem System / ähnliche Projekte zurückzugreifen
- TA sollte v.a. mit Benutzern und anderen Experten des Geschäftsbereichs eng zusammenarbeiten, um Bereiche mit Geschäftsrisiken zu bestimmen welche beim Testen berücksichtigt werden sollten
- TA kann auch besonders hilfreich sein um potentielle Ausw. von Risiken auf Benutzer & Betroffene zu identifizieren
- Beispiele von Risiken, die beim Projekt identifiziert werden könnten:
    - Probleme mit Genauigkeit der SW-Funktionalität, z.B. falsche Berechnungen
    - Benutzbarkeitsprobleme, z.B. zu wenig Tastaturkürzel
    - Erlernbarkeitsprobleme, z.B. keine Anweisungen für Benutzer an Hauptentscheidungspunkten

2.4.3. Risikobewertung (=Risikoanalyse!)

- Risikoanalyse = Untersuchung der identifizierten Risiken
- Kategorisierung: Eintrittswahrscheinlichkeit (Bewertung technisches Risiko) & Auswirkungen (Bew. Geschäftsrisiko)

- Eintrittswahrscheinlichkeit = Wahrscheinlichkeit, dass das potentielle Problem vorhanden ist → technisches Risiko
- Technical Test Analyst → Mitwirkung bei Festlegung der Risikostufe der potentiellen technischen Risiken
- Test Analyst → Mitwirkung bei potentiellen geschäftlichen Auswirkungen des Problems, falls es auftritt
- Auswirkung bei Risikoeintritt = Ausmass der Wirkung auf Benutzer, Kunden, &/ andere Betroffene → Geschäftsrisiko
- Folgende Faktoren beeinflussen Geschäftsrisiken:
  - Häufigkeit der Nutzung der betroffenen Funktion
  - Entgangene Geschäfte
  - Mögliche finanzielle, ökologische oder soziale Verluste oder Haftungsansprüche
  - Zivil- oder strafrechtliche Massnahmen
  - Sicherheitsbedenken
  - Geldstrafen, Aberkennung der Lizenz
  - Keine vernünftigen Lösungsalternativen
  - Sichtbarkeit des Funktionsmerkmals
  - Negatives Erscheinungsbild, wenn Mängel bekannt werden, Negativschlagzeilen, Reputationsschaden
  - Verlust von Kunden
- TM gibt Richtlinie vor, TA legt Risikostufen für Geschäftsrisiken fest → hoch, mittel, gering / mit Zahlen
- Wenn Risiko nicht objektiv anhand definierten Skala angegeben werden kann → keine echte quantitative Messung
- Auch Messung von Wahrscheinlichkeit & finanzieller Kosten/Konsequenzen i.d.R. sehr schwierig
  → meist qualitative Bestimmung der Risikostufe, trotzdem oft mit Zahlen untermauert
- Bsp.: TM gibt vor, Geschäftsrisiko mit Wert 1 (hoch) -10 (tief) zu kategorisieren
  → Werte für Eintrittswahrsch. & Auswirkung miteinander multiplizieren → Kennzahl für Gesamtrisiko → Rating für Priorisierung der Risikoreduzierungsaktivitäten
- Bei einigen risikoorientierten Testvorgehensweisen (z.B. PRISMA) → keine Kombination
  → getrennte Behandlung von technischen & geschäftlichen Risiken

2.4.4. Risikobeherrschung

- TA-Aufgaben im Projekt:
  - Produktrisiken reduzieren via Testfälle &/ Reviews der SW-Artefakte (z.B. Anforderungen, Entwürfe, Benutzerdoku)
  - Geeignete Massnahmen zur Risikobeherrschung umsetzten (festgelegt in Teststrategie / Testkonzept)
  - Bekannte Risiken neu bewerten, basierend auf im Projektverlauf zusätzlich verfügbare Informationen
  - Neue Risiken erkennen, welche durchs Testen hervortreten
- Testen = eine Art der Risikobeherrschung für Produkt- und Qualitätsrisiken
- Wird Fehler gefunden = Risikoreduktion, da Bewusstsein für Fehler höher wird und Möglichkeit vorhanden, diese vor Systemfreigabe zu entfernen
- Wird kein Fehler gefunden = Risikoreduktion, da Beweis, dass System unter getesteten Bedingungen funktioniert

- TA → untersuchen Möglichkeiten, genaue Testdaten zu sammeln, erstellen & testen realistische Szenarien, leiten / überwachen Benutzbarkeitsstudien

2.4.4.1. Tests priorisieren

- Risikostufe → Verwendung bei Testpriorisierung
- Bsp. 1: TA stellt fest, dass Transaktionsgenauigkeit eines Buchhaltungssystems hohes Risiko gibt. Um dieses Risiko zu beherrschen, arbeitet Tester mit Geschäftsbereichsexperten zusammen, um solide Datenmenge zusammenzutragen, die verarbeitet werden kann, sodass Genauigkeit verifizierbar wird
- Bsp. 2: TA stellt fest, dass Probleme mit Benutzung = bedeutendes Risiko → Anstatt Benutzer-Abnahmetests abzuwarten: hohe Prio setzen auf frühe Benutzbarkeitstests während Integrationstestphase
- Manchmal: Testen von hohen Risikostufen vor niedrigeren Risikostufen → Depth-first = Testen in die Tiefe
- Andere Fälle: Stichproben für alle Risikostufen, Risikogewichtung & Testauswahl, sodass jedes Risiko mind. 1x abgedeckt wird → Breadth-first = Testen in die Breite
- Unabhängig ob in Tiefe oder Breite → Testzeit kann ablaufen, ohne fertig zu sein → Info an Management über verbleibendes Risiko → Management kann entscheiden, ob weiter getestet werden soll oder ob das Restrisiko an Benutzer, Kunden, Helpdesks oder technische Unterstützung &/ an das Bedienpersonal weitergegeben wird

2.4.4.2. Anpassung des Testens an weitere Testzyklen

- Risikobewertung = fortlaufender Prozess. Alle weiteren geplanten Testzyklen sollten auf Basis einer neuen Risikoanalyse erfolgen, unter Berücksichtigung folgender Faktoren:
  - Mögliche neue / deutlich veränderte Produktrisiken
  - Instabile / fehleranfällige Systembereiche, identifiziert im Laufe des Testens
  - Risiken als Folge behobener Fehler
  - Fehler, die sich beim Testen als typische Fehler herausgestellt haben
  - Unzureichend getestete Systembereiche → Bereiche mit geringer Testüberdeckung

3. Testverfahren

3.1. Einführung

- Einteilung der Testentwurfsverfahren nach folgenden Kategorien:
  - Spezifikationsorientiert (→ verhaltensbasiert / Black-Box)
  - Fehlerbasiert
  - Erfahrungsbasiert
- Die Verfahren ergänzen sich gegenseitig & können für jede Testaktivität unabhängig der Teststufe eingesetzt werden
- Alle 3 Kategorien für funktionale & nicht-funktionale Qualitätsmerkmale verwendbar

- Hier behandelt: primär Fokus auf Festlegung optimaler Testdaten (z.B. Äquivalenzklassen) oder auf Ableitung von Testabläufen (z.B. Zustandsmodelle)
- Es ist üblich, die Testfallerstellungsverfahren zu kombinieren

3.2. Spezifikationsorientierte Testverfahren

- Spezifikationsorientierte Testverfahren: Anwendung auf Testbedingungen, um Testfälle basierend auf Analyse der Testbasis für Komponente / System abzuleiten ohne dabei deren interne Struktur zu berücksichtigen
- Gemeinsames Merkmal: Während Testentwurf werden Modelle erstellt (z.B. Zustandsübergangsdiagramme & Entscheidungstabellen). Daraus werden die Testbedingungen systematisch abgeleitet
- Einige Verfahren: Kriterien für Überdeckungsgrad als Mass für Testentwurf & -durchführungsaufgaben: Wen Kriterien für Überdeckungsgrad vollständig erfüllt, heisst das nicht, dass die Menge an Tests vollständig ist, sondern vielmehr, dass das Modell keine weiteren Tests vorschlägt um Überdeckung zu erhöhen
- Spezifikationsorientierte Tests basierend meist auf Anforderungsdokumentation für das System
- Anforderungsspezifikation → beschreibt das Systemverhalten, v.a. hinsichtlich seiner Funktionalität → daraus lassen sich Tests ableiten, die das Systemverhalten prüfen
- Z.T. sind die Anforderungen nicht dokumentiert (z.B. bei Ersatz einer Funktion) und liegen nur implizit vor

3.2.1. Äquivalenzklassenbildung (ÄK-Bildung)

- Einleitung: Wird eingesetzt um #Testfälle zu reduzieren, die erforderlich sind, um die Verarbeitung von Eingabe- und Ausgabewerten, interner und zeitbezogener Werte effektiv zu testen. Durch Aufteilung (→ Partitioning) erstellte Äquivalenzklassen (→ Äquivalenzpartitionen) gibt es Wertemengen, die auf gleicher Weise bearbeitet werden → 1 repräsentativer Wert wird ausgewählt, gilt als Überdeckung aller Werte derselben Äquivalenzklasse
- Anwendbarkeit: Auf allen Teststufen. Dann geeignet, wenn SW die Werte einer Klasse/Partition auf selbe Weise verarbeitet und diese Werte durch die Anwendung nicht interagieren. Klassen für gültige sowie ungültige Daten erstellbar. Am nützlichsten in Kombination mit Grenzwertanalyse → Einschluss der Werte im Bereich der Grenzen. Sehr gebräuchlich für Smoke-Test eines neuen Builds/Releases → Schnell feststellbar, ob grundlegende Funktionalität erfüllt ist
- Einschränkungen/Schwierigkeiten: Falls Werte gleicher Klasse dennoch unterschiedlich verarbeitet werden → Fehler können übersehen werden. Zudem: Äquivalenzklassen müssen sorgfältig ausgewählt werden. Bsp: Eingabefeld für positive & negative Zahlen → 2 Äquivalenzklassen bilden (→ pos. & neg.). 0 zulässig oder nicht? → weitere Äquivalenzklasse für 0! → TA muss die Verarbeitungsprozesse der SW verstehen.
- Überdeckung: # überdeckten ÄK : # gesamter ÄK → Falls mehrere Werte gleicher ÄK: kein Überdeckungsmehrwert
- Fehlerarten: Funktionale Fehlerzustände bei Verarbeitung verschiedener Datenwerte

3.2.2. Grenzwertanalyse (GWA)

- Einleitung: Wird eingesetzt um Werte an Grenzen der eingeteilten ÄK zu testen. 2 Vorgehensweisen: Testen mit 2 / 3 Werten (→ Grenzen = Min./Max.-Werte der ÄK). Ob 2 oder 3 Werte genommen werden orientiert sich am Risiko.
  Bsp.: ÄK = Werte von 1-10, Inkrement = 0.5
  2 Werte: Obere Grenze: [10, 10.5], Untere Grenze: [0.5, 1]
  3 Werte: Obere Grenze: [9.5, 10, 10.5], Untere Grenze: [0.5, 1, 1.5]
- Anwendbarkeit: Auf allen Teststufen. Dann geeignet, wenn Elemente der ÄK durch Ordnungsrelation (→ «abgrenzbare Wertebereiche») vergleichbar sind. Bsp.: Zahlenbereich = geordnete ÄK (=! Z.B. Rechteckige Objekte)
- Einschränkungen/Schwierigkeiten: Vgl: ÄK: Falsche Einteilung = Fehler können übersehen werden. Kleinste Inkremente müssen klar sein. Durch GWA können nur geordnete ÄK getestet werden, die beschränken sich jedoch nicht auf eine Gruppe gültiger Eingaben. Bsp.: Anzahl Zellen in EXCEL. ÄK = Alle Zellen bis maximale Anzahl, inkl. Zellen an oberer Grenze. Nächste ÄK beginnt dagegen mit Zelle genau über der Grenze.
- Überdeckung: # getesteter Grenzwerte : # insgesamt identifizierter Grenzwerte (mit 2- / 3-Wert-Methode)
- Fehlerarten: Aufdeckung verschobener oder fehlender Grenzen, es können auch zusätzliche Grenzen gefunden werden. Aufdeckung von Fehlern in Zusammenhang mit Verarbeitung von Grenzwerten, insbesondere bei logischen Bedingungen mit «kleiner als» & «grösser als» (→ wenn Grenzverschiebungen vorliegen). Auch nicht-funktionale Fehlerzustände sind aufdeckbar, z.B. bei Abweichungen bei den Lastgrenzen, Bsp.: System unterstützt 1000 gleichzeitige Benutzer

3.2.3. Entscheidungstabellen (ET)

- Einleitung: Wird eingesetzt um Zusammenwirken zwischen Kombinationen von Bedingungen zu testen. Liefern geeignete Methode, um zu verifizieren, dass alle vorhandenen Kombinationen von Bedingungen, Beziehungen und Beschränkungen berücksichtigt und getestet werden. ET können schnell sehr gross werden → intelligente Auswahl der interessanten Kombinationen = reduzierter ET-Test, Reduktion der Anzahl Kombinationen auf jene mit unterschiedlichen Ausgaben, redundante & unrealistische Tests werden weggelassen. Ob vollständiger oder reduzierter ET-Test gemacht wird = f(Risiko)
- Anwendbarkeit: Integrations-, System- und Abnahmetest. Modultest ist abhängig vom Code, falls dieser für Entscheidungslogik zuständig ist. Besonders nützlich, wenn Anforderungen in Form von Ablaufdiagrammen / Tabellen von Geschäftsregeln vorliegen. ET sind auch zur Spezifikation von Anforderungen anwendbar, daher liegen die Anforderungsspezifikationen manchmal bereits in dieser Form vor, z.T. nur in Textform, aber ableitbar. Beim Testentwurf müssen jedoch auch die nicht ausdrücklich spezifizierten Kombinationen von Bedingungen abgedeckt werden. Nur wenn wirklich alle möglichen Kombinationen berücksichtigt werden, sind ET ein gutes Testentwurfswerkzeug
- Einschränkungen/Schwierigkeiten: Identifikation sämtlicher Kombinationen ist schwierig, v.a. wenn Anforderungen nicht gut spezifiziert / inexistent sind. Oft wird eine Menge von Bedingungen erstellt, nur um dann festzustellen, dass das erwartete Ergebnis unbekannt ist

- Überdeckung: Minimale Überdeckung = 1 Test / Spalte. Setzt voraus, dass es keine Mehrfachbedingungen gibt, und dass in der Spalte alle möglichen Kombinationen von Bedingungen berücksichtigt werden. Bei Auswahl der Tests aus ET: Berücksichtigung von etwaigen Grenzbedingungen nötig → # Testfälle kann sich dadurch erhöhen. ET wird ergänzt durch ÄK-Bildung & GWA.
- Fehlerarten: Inkorrekte Verarbeitung, die auf bestimmte Kombinationen von Bedingungen basiert und zu unerwarteten Ergebnissen führt. Beim Erstellen der ET können Fehlerzustände in Specs aufgedeckt werden (meistens Lücken oder Widersprüche). Beim Testen: Probleme mit Kombinationen von Bedingungen aufdeckbar, die gar nicht / nicht gut gehandhabt werden

3.2.4. Ursache-Wirkungs-Graph-Analyse (UWG-A)

- Einleitung: UWG können aus Quellen abgeleitet werden, wo die Funktionslogik (→ «Regeln») eines Programms beschrieben ist (→ User-Stories, Ablaufdiagramme). Geben graphischen Überblick über logische Struktur eines Programms und dienen meist als Basis für die Erstellung von ET. Durch Ermittlung von Entscheidungen in Form von UWG &/ ET → systematische Testüberdeckung der SW-Logik
- Anwendbarkeit: Integrations-, System- und Abnahmetest. Modultest ist abhängig vom Code, falls dieser für Entscheidungslogik zuständig ist. UWG zeigt Kombinationen von auslösenden Bedingungen, die zu Ergebnissen führen (Kausalität):
    - Kombinationen von Bedingungen, die Ergebnisse ausschlissen («nicht»)
    - Mehrfachbedingungen, die «wahr» sein müssen, um zu einem Ergebnis zu führen («und»)
    - Alternative Bedingungen, die «wahr» sein müssen, um zu einem bestimmten Ergebnis zu führen («oder»)

  UWG ist einfacher als Beschreibung in Textform
- Einschränkungen/Schwierigkeiten: Fürs Erlernen der UWG-A ist mehr Zeit & Aufwand erforderlich. Werkzeugunterstützung notwendig. UWG haben spezifische Notation, die Ersteller & Leser verstehen soll
- Überdeckung: Minimale Überdeckung = jede mögliche Ursache testen, inkl. Kombinationen von auslösenden Bedingungen, die mit einer Wirkung in Verbindung gesetzt ist. In UWG können auch Beschränkungen in Zusammenhang mit Daten / Ablauflogik spezifiziert werden
- Fehlerarten: Inkorrekte Verarbeitung, die auf bestimmte Kombinationen von Bedingungen basiert und zu unerwarteten Ergebnissen führt. Durch UWG kann der geforderte Detaillierungsgrad in der Testbasis erreicht werden → Verbesserung der Granularität & Qualität der Testbasis → hilft den Testern bei Identifizierung fehlender Anforderungen

3.2.5. Zustandsbasierter Test (ZB Test)

- Einleitung: Wird eingesetzt um SW in Bezug auf ihre definierten Zustände und deren Übergänge zu testen. Zustandsübergänge können gültig / ungültig sein. Ereignisse sind Auslöser für die SW, von einem Zustand in einen anderen zu wechseln & Aktionen auszuführen. Ereignisse sind von Bedingungen beeinflusst (→ Schutzbedingungen / Übergansschutz), was sich wiederum auf den Zustandspfad auswirkt. Bsp.: Einloggen mit gültigen UN & PW = anderer Zustandsübergang als Einloggen mit ungültigem PW.

Zustandsübergänge werden entweder in Zustandsdiagrammen (→ alle gültigen Zustandsübergänge) oder in Zustandstabellen (→ gültige & ungültige Zustandsübergänge) dargestellt.

- Anwendbarkeit: Auf allen Teststufen. Für jede SW, die definierte Zustände hat, die durch Ereignisse ändern (z.B. Wechsel des Bildschirms). Prädestiniert: eingebettete SW, browserbasierte SW, Transaktionssysteme, Steuerungssysteme (→ Verkehrsampelsystemsteuerung)
- Einschränkungen/Schwierigkeiten: Schwierig: Bestimmung der Zustände. Wenn SW GUI hat, sind häufig die einzelnen Bildschirmmasken = Zustände. Bei eingebetteter SW: Zustände = f(HW). Basisgrösse = einzelner Zustandsübergang (→ 0-Switch) → Fehlerrate vorhanden, aber kleiner als beim Testen von Transaktionenfolgen. Sequenz aus 2 aufeinander folgender Übergänge = 1-Switch, Sequenz aus 3 aufeinander folgender Übergänge = 2-Switch, usw. → Bezeichnung oft auch als N-1-Switch (N = #Zustandsübergänge) → 1x Übergang = 1-1-Switch ☺
- Überdeckung: Minimale Überdeckung = Jeder Zustand & jeder Übergang 1x testen. 100% Zustandsübergansüberdeckung (→ 100% 0-Switch-Überdeckung / 100% logische Zweigüberdeckung) garantiert, dass jeder Zustand getestet wird, vorausgesetzt der Systementwurf / Zustandsübergansmodell ist korrekt. Je nach Beziehungen der Zustände & Übergänge kann es sein, dass einige Übergänge mehrmals ausgeführt werden müssen, um andere Übergänge wenigstens 1x auszuführen. N-Switch-Überdeckung bezieht sich auf Sequenzen mit >1 Zustandsübergängen. Bsp.: 100% 1-Switch-Überdeckung → jede gültige Sequenz von 2 aufeinander folgender Zustandsübergängen muss mindestens 1x getestet werden → höhere Fehlerfindungsquote als mit 100% 0-Switch. Rundreise-Überdeckung → Folgen von Zustandsübergängen, die Schleifen formen → 100% Rundreise-Überdeckung = alle Zustände ab einem beliebigen Punkt und zurück getestet. Muss für alle Zustände getestet werden, die zur Schleife gehören. Vollständige Überdeckung: Alle möglichen Sequenzen mit Länge N-1 getestet (N = #Zustände). Durch Einbezug der ungültigen Übergänge → höhere Überdeckung. Aus Überdeckungsanforderungen und abzudeckenden Zustandsübergängen muss hervorgehen, ob ungültige Zustandsübergänge beinhaltet sind
- Fehlerarten: Inkorrekte Verarbeitungen im aktuellen Zustand als Folge der Verarbeitung in einem vorangegangenen Zustand, falsch / nicht unterstützte Zustandsübergänge, Zustände ohne Ausgang, Bedarf an Zuständen / -übergängen die nicht existieren. Bei Erstellung des Zustandsmodells → Aufdeckung von Defekten in Specs. Meistens Lücken oder Widersprüche

3.2.6. Kombinatorische Testverfahren (K Testverfahren)

- Einleitung: Wird eingesetzt, wenn SW mit mehreren Parametern getestet wird, von denen jeder mehrere Werte hat (→ kaum Zeit für alle Test vorhanden, daher Auswahl nötig). Parameter müssen unabhängig und soweit kompatibel sein, dass jede Option für jeden beliebigen Faktor mit jeder Option eines beliebigen anderen Faktors kombiniert werden kann. Bei Klassifikationsbäumen ist es zulässig, einige Kombinationen auszuschliessen, wenn bestimmte Optionen inkompatibel sind → daraus ist nicht ableitbar, dass sich die kombinierten Faktoren nicht gegenseitig beeinflussen, jedoch sollte das in akzeptabler Art und Weise erfolgen. Durch K-Testen lässt sich eine geeignete Teilmenge dieser Kombinationen identifizieren um ein vorgegebenes Überdeckungsmass zu erzielen. # Elemente für diese Kombination wird vom TA

bestimmt → umfasst einzelne oder mehrere Elemente. Verschiedene unterstützende Werkzeuge vorhanden: entweder Auflistung der Parameter und deren Werte (→ Paarweises Testen & Orthogonale Arrays) oder grafische Darstellung (→ Klassifikationsbäume). Paarweises Testen = Verfahren, bei dem Paare von Variablen in Kombination getestet werden.

  - Paarweises Testen: Verfahren, bei dem Paare von Variablen in Kombination getestet werden
  - Orthogonale Arrays: Vordefinierte, mathematisch genaue Tabellen, die dem TA ermöglichen, die zu testenden Elemente durch Variablen aus dem Array zu ersetzen → resultiert in einer Kombinationsmenge, die ein Überdeckungsmass erzielen.
  - Klassifikationsbäume: Ermöglichen dem TA, die Grösse der zu testenden Kombinationen zu definieren → Kombination von 2, 3, etc. Werten

- Anwendbarkeit: Integrations-, System- und Systemintegrationstests. Zu viele Kombinationen von Parametern gibt es in mind. Folgenden 2 Situationen, wobei in beiden Fällen kombinatorisches Testen eingesetzt werden kann, um eine Teilmenge von Kombinationen in akzeptablem Umfang zu identifizieren:
  1. Testfälle mit mehreren Parametern mit jeweils mehreren möglichen Werten, z.B. Bildschirmmaske mit mehreren Eingabefeldern → Kombinationen von Parameterwerten = Eingabedaten für die Testfälle
  2. Systeme die in einer Reihe von Dimensionen konfigurierbar sind → führt zu einem potentiell grossen Konfigurationsraum

  Für Parameter mit sehr vielen Werten → ÄK-Bildung oder anderer Auswahlmechanismus, welcher zunächst auf jeden Parameter einzeln angewendet wird, um die Anzahl der Werte für die einzelnen Parameter zu reduzieren. Anschliessend K-Testen
- Einschränkungen/Schwierigkeiten: Grösste Einschränkung = Annahme, dass Ergebnisse einiger weniger Tests repräsentativ sind für alle Tests & das diese wenigen Tests die erwartete Nutzung abbilden. Falls unerwartete Interaktion zwischen bestimmten Variablen, wird diese mit dieser Testart möglicherweise nicht entdeckt, sofern diese bestimmte Kombination nicht getestet wird.
  Diese Verfahren können Personen ohne technischen Hintergrund nur schwer erklärt werden da diese die Logik der Reduzierung der Tests nicht verstehen.
  Identifizierung der Parameter & deren Werte ist schwierig. Auch schwierig die minimale Anzahl Kombinationen für bestimmtes Überdeckungsmass manuell zu bestimmen → dafür werden oft Werkzeuge eingesetzt, einige davon bieten die Möglichkeit, bestimmte Kombinationen zwingend ein- resp. auszuschliessen → Gewichtung von Faktoren für wichtige Geschäftsbereichen /& Produktverwendungen möglich
- Überdeckung: Mehrere Überdeckungsmasse vorhanden:
  Niedrigste = 1-fache / Singleton-Überdeckung → jeder Wert eines jeden Parameters muss in mindestens einer der ausgewählten Kombinationen vorhanden sein.
  Nächstes = 2-fache / Paarweise Überdeckung → jedes Wertepaar von zwei beliebigen Parametern muss in mindestens einer Kombination vorhanden sein.
  Allgemein = n-fache Überdeckung → jede Teilkombination von Werten einer jeden Menge von n Parametern muss in der Menge der ausgewählten Kombinationen enthalten sein.
  → Je höher n, desto mehr Kombinationen nötig, um 100% Überdeckung zu erreichen
  → Mindestüberdeckung bei diesen Verfahren = 1 Testfall pro Kombination, die vom Werkzeug erstellt wird

- Fehlerarten: Fehlerzustände in Zusammenhang mit den kombinierten Werten mehrerer Parameter

3.2.7. Anwendungsfallbasierter Tests (AFB Test)

- Einleitung: Liefert transaktionale, auf Szenarien basierte Tests, welche die Benutzung des Systems nachahmen. Ein Anwendungsfall beschreibt die Interaktionen zwischen Aktoren & System, die ein Ergebnis erzielen. Aktoren = Benutzer / externe Systeme
- Anwendbarkeit: System- und Abnahmetest, je nach Integrationsebene auch in Integrationsstufe, je nach Verhalten der zu testenden Komponente auch bei Komponententest. AFB Tests häufig auch Basis für Performanztest, da sie die Anwendung des Systems realistisch abbilden. Die Szenarien können virtuellen Benutzern zugewiesen werden, um realistische Systemlast zu erzeugen
- Einschränkungen/Schwierigkeiten: AF müssen realistische Benutzertransaktionen abbilden, die Info dazu stammt von Benutzer / -vertreter → Wert des AF verringert sich, wenn dieser die Aktivitäten der echten Benutzer nicht genau wiedergibt. Genaue Spezifikation der verschiedenen alternativen Pfade (→ Abläufe) unerlässlich. AF sollen als Richtlinien angesehen werden, nicht als vollständige Definition dessen was zu testen ist → u.U. liefern diese keine eindeutige Definition der gesamten Menge von Anforderungen. Erstellung von anderen Modellen (z.B. Ablaufdiagramm) aus Schilderung des Anwendungsfalls empfehlenswert → Verbesserung der Testgenauigkeit
- Überdeckung: Mindestabdeckung = 1 Testfall für (positiven) Hauptpfad, je ein Testfall für jeden Alternativpfad / -prozessablauf. Alternativpfad kann auch Ausnahme- oder Ausfallpfad sein → können auch Verlängerung des Hauptpfades sein. Prozentuale Überdeckung = #getesteter Pfade : #gesamthaft vorhandener Haupt- und Alternativpfade
- Fehlerarten: Fehlerhafte Verarbeitung spezifizierter Szenarien, versäumte Verarbeitung alternativer Pfade, inkorrekte Verarbeitung der vorliegenden Bedingungen, umständliches / inkorrektes Berichten von Fehlern

3.2.8. User Story-basiertes Testen (US Testen)

- Einleitung: Bei agilen Methodologien wie z.B. Scrum → Anforderungen = User Stories = kleine Funktionseinheiten, die in einer einzigen Iteration entworfen, entwickelt, getestet und vorgeführt werden können. In diesen US ist eine Beschreibung der zu implementierenden Funktionalität enthalten, alle nicht-funktionalen Kriterien sowie Abnahmekriterien, die erfüllt sein müssen, damit die US als vollständig betrachtet werden kann
- Anwendbarkeit: Auf allen Teststufen. Überwiegend in agilen sowie ähnlich ausgeprägtem, iterativem & inkrementellem Umfeld. Für funktionales & nicht-funktionales Testen. Es wird erwartet, dass der Entwickler die zur US entwickelte Funktionalität demonstriert, bevor der Code an das Team zur Durchführung der nächsten Stufe der Testaufgaben (z.B. Integrations- oder Performanztest) übergeben wird
- Einschränkungen/Schwierigkeiten: US = kleine Inkremente der Funktionalität → manchmal notwendig, Treiber und Platzhalter zu erstellen, damit die gelieferte Teilfunktionalität überhaupt testbar ist → Programmierfähigkeiten & Benutzung von Werkzeugen notwendig, die das Testen unterstützen, wie z.B. API-Werkzeuge. Erstellung der Treiber & Platzhalter meist Aufgabe des Entwicklers, kann aber auch TTA

machen. Falls - wie bei den meisten agilen Projekten – kontinuierliche Integration erfolgt, wird der Bedarf von Treibern und Platzhaltern minimiert
- Überdeckung: Mindestüberdeckung einer US = alle spezifizierten Abnahmekriterien erfüllt
- Fehlerarten: Funktionale Fehlerzustände, Fehlerzustände in Zusammenhang mit der Integration der Funktionalität der neuen US mit der bereits vorhandenen Funktionalität. Da US unabhängig entwickelt werden können, gibt es potentielle Fehler bei Performanz, Schnittstellen und Fehlerbehandlungen. Wichtig, dass TA nicht nur die neuen Funktionen testet, sondern bei jeder US Freigabe auch einen Integrationstest macht

3.2.9. Wertebereichsanalyse (WB Analyse)

- Einleitung: Domain / WB = definierte Menge von Werten. Es kann ein WB einer einzelnen Variablen sein (→ eindimensionaler WB, z.B. Männer über 24 und unter 66 Jahren) oder um WB interagierender Variablen (→ multidimensionaler WB, z.B. Männer, 24 - 66-Jährig UND Gewicht = 69 – 90 kg) → Jeder Testfall für multidimensionaler WB muss geeignete Werte für jede der betroffenen Variablen enthalten. Für 1D-WB Analyse → normalerweise via ÄK & GWA. Wenn ÄK definiert, wählt TA Werte für jede der ÄK: ein Wert innerhalb Wertebereich (IN), einen ausserhalb (OUT), einen auf der Grenze (ON), einen knapp neben der Grenze (OFF) → jeder Teilbereich sowie seine jeweiligen Grenzbedingungen werden getestet. Bei multidimensionalen WB → #Testfälle steigt exponentiell mit #betroffener Variablen → Vorgehen die auf der Domain-Theorie basiert führt hingegen zu lineare, Zuwachs. Formale Vorgehensweise beinhaltet eine Theorie der Fehlerzustände (→ Fehlermodell), das bei ÄK und GWA fehlt → mit kleinerer Testmenge werden Fehlerzustände aufgedeckt, die bei grösserer, heuristischer Testmenge wahrscheinlich unentdeckt bleiben würden. Bei Tests multidimensionaler Wertebereiche wird das Testmodell als Entscheidungstabelle oder Wertebereichs-Matrix erstellt → alles das mehr als 3D ist, braucht i.d.R. Computerunterstützung
- Anwendbarkeit: Auf allen Teststufen, am Häufigsten für Integrations- und Systemtest. WB Analyse = Kombination von ET, ÄK, GWA, um mit kleinerer Testmenge die wichtigen Bereiche abzudecken und in welchen Fehlerwirkungen wahrscheinlich sind. WB Analyse wird oft bei sehr vielen potenziell miteinander interagierenden Variablen angewendet, wenn ET nicht handhabbar sind.
- Einschränkungen/Schwierigkeiten: Für gründliche WB Analyse → gutes SW Verständnis nötig. Wird ein WB übersehen → Testen kann unzureichend sein. Jedoch ist Wahrscheinlichkeit hoch, dass WB entdeckt wird, da die OFF- und OUT-Variablen möglicherweise in den unentdeckten Bereich fallen. WB Analyse = starkes Verfahren, besonders wenn Tester & Entwickler zusammenarbeiten um die Testbereiche festzulegen
- Überdeckung: Mindestüberdeckung = in jedem WB jeder IN-, OUT-, ON-, und OFF-Wert im Test abgedeckt. Falls sich Werte überschneiden (z.B. OUT-Wert eines WB = IN-Wert eine anderen WB) reicht 1 Test. In Praxis daher oft weniger als 4 Tests pro WB
- Fehlerarten: Funktionale Probleme innerhalb WB, Behandlung von Grenzwerten, Probleme bei Interaktion von Variablen, Fehlerbehandlung (besonders für Werte die nicht zu einem gültigen WB gehören)

3.2.10. Kombination von Verfahren

- Manchmal werden Verfahren kombiniert, z.B. mit ET identifizierte Bedingungen werden ÄK-Bildung unterzogen, um unterschiedliche Möglichkeiten herauszufinden, wie eine Bedingung erfüllt werden kann
- Tests decken dann nicht nur alle Kombinationen von Bedingungen ab, sondern es werden zusätzliche Tests für die ÄK der Bedingungen erstellt, sodass auch diese überdeckt werden
- Bei Auswahl des geeigneten Testverfahrens sollte der TA die Anwendbarkeit des Verfahrens, die Einschränkungen und Schwierigkeiten sowie die Testziele bezüglich Überdeckung und der aufdeckbaren Fehlerzustände berücksichtigen
- Möglicherweise gibt es für eine Situation nicht ein einziges «bestes» Verfahren. Oft liefert eine Kombination verschiedener Verfahren die grösstmögliche Überdeckung – vorausgesetzt für die korrekte Anwendung der Verfahren ist ausreichend Zeit und fachliche Kompetenz vorhanden

3.3. Fehlerbasierte Verfahren

3.3.1. Verwendung von fehlerbasierte Verfahren (FB Verfahren)

- Einleitung: Bei FB-Testentwurfsverfahren → Art des gesuchten Fehlers = Basis für Testentwurf, wobei Tests systematisch davon abgeleitet werden, was über die Fehlerart bekannt ist. Tests werden aus Fehlertaxonomien (→ kategorisierte Listen) abgeleitet, die völlig unabhängig von der SW sein können. Taxonomien können Listen möglicher Fehlerarten, Grundursachen, Symptome von Fehlerwirkungen u.a. sein. Beim FB Testen können auch Listen mit den identifizierten Risiken & -szenarien als Grundlage dienen. Ermöglicht dem Tester auf besondere Arten von Fehlern abzuzielen, oder eine Fehlertaxonomie mit bekannten & häufigen Fehlerzuständen bestimmter Arten systematisch durchzuarbeiten. TA nutzt Taxonomiedaten, um Testziel zu bestimmen (→ bestimmte Fehlerart finden). TA erstellt aufgrund dieser Info Testfälle und Testbedingungen, die den Fehlerzustand aufdecken, sofern vorhanden
- Anwendbarkeit: Auf jeder Teststufe, meistens auf Systemstufe. Standardtaxonomien vorhanden, passend zu mehreren SW-Arten. Diese Testart ist nicht produktspezifisch, unterstützt auch branchenspezifisches Standardwissen. Bei Gebrauch von branchenspezifischen Taxonomien → Metriken projekt- &/ organisationsübergreifend verfolgbar
- Einschränkungen/Schwierigkeiten: Verschiedene Fehlertaxonomien vorhanden, z.B. mit Schwerpunkt Benutzbarkeitstest. Anwendbare Taxonomie finden & auswählen wichtig. Für innovative SW → kaum Taxonomien vorhanden. Einige Unternehmen haben eigene Taxonomien, mit wahrscheinlichen / häufigen Fehlern. Vor Testbeginn muss die erwartete Überdeckung festgelegt werden
- Überdeckung: Dieses Verfahren liefert auch Kriterien für Überdeckungsgrad, der bestimmt, wann alle sinnvollen Tests identifiziert wurden. In Praxis sind Kriterien für Überdeckungsgrade bei fehlerbasierten Testentwurfsverfahren weniger systematisch als bei spezifikationsorientierten Verfahren. Vorgabe von lediglich allgemeinen Regeln zur Überdeckung. Liegt im eigenen Ermessen, wo Grenze zur sinnvollen Überdeckung im Testentwurf liegt. Kriterium für Überdeckungsgrad bedeutet nicht, dass Testmenge

vollständig ist. Gibt lediglich an, dass dieses Testverfahren keine weiteren Tests mehr vorschlägt

- Fehlerarten: Hängen gewöhnlich von Taxonomie ab. Z.B. bei Benutzerschnittstellen-Taxonomie → haupts. Benutzerschnittstellenspezifisch, andere Fehler als Nebenprodukt auffindbar

3.3.2. Fehlertaxonomien (FT)

- FT sind kategorisierte Listen von Fehlerarten. Allgemeine (abstrakte Richtlinien) oder Spezifische.
  Bsp.: Taxonomie für mögliche Fehler von Benutzerschnittstellen → können allgemeine Punkte enthalten, z.B. Funktionalität, Fehlerbehandlung, grafische Darstellung, Performanz.
  Detaillierte Taxonomie kann Liste mit allen möglichen Objekten von Benutzerschnittstellen enthalten, bes. bei grafischen Benutzeroberflächen, und unsachgemässe Behandlung dieser Objekte benennen, z.B.:
  - Textfelder
    - Gültige Daten werden nicht übernommen
    - Ungültige Daten werden übernommen
    - Länge der Eingabe wird nicht verifiziert
    - Sonderzeichen werden nicht erkannt
    - Fehlermeldungen sind nicht informativ
    - Benutzer kann fehlerhafte Daten nicht korrigieren
    - Regeln werden nicht angewendet
  - Datumsfeld
    - Gültige Datumsangaben werden nicht übernommen
    - Ungültige Datumsangaben werden nicht abgewiesen
    - Datumsbereiche werden nicht verifiziert
    - Präzisionsdaten werden nicht korrekt behandelt (zum Beispiel hh:mm:ss)
    - Benutzer kann fehlerhafte Daten nicht korrigieren
    - Regeln werden nicht angewendet (z.B. Enddatum muss größer sein als Startdatum)
- Viele Taxonomien vorhanden, gekaufte oder von Unternehmen selber erstellte für bestimmte Zwecke. Intern entwickelte → spezifische Fehleraufdeckung, welche innerhalb des Unternehmens häufig gefunden werden. Bevor Taxonomie erstellt wird: Zielsetzung der Taxonomie festlegen. Bsp.: Probleme mit GUI, die in Produktivsystemen entdeckt wurden / Probleme mit Behandlung von Eingbefeldern
- Schritte zur Erstellung einer Taxonomie:
  1. Ziel und gewünschten Detaillierungsgrad definieren
  2. Vorhandene Taxonomie als Basis auswählen
  3. Werte & häufige Fehlerzustände spezifizieren, die in Unernehmung &/ ausserhalb in Praxis vorgekommen sind
- Je detaillierter, desto höher der Zeitaufwand für Entwicklung & Wartung der Taxonomie
- Hoher Detaillierungsgrad = höhere Reproduzierbarkeit der Testergebnisse
- Detaillierte Taxonomien können redundant sein → ermöglichen dem Tester jedoch die Aufteilung des Testens, ohne Verlust an Informationen / Überdeckung
- Nachdem geeignete Taxonomie ausgewählt wurde, kann diese für Entwurf von Testbedingungen / Testfällen dienen
- Risikoorientierte Taxonomie kann Fokus auf bestimmen Risikobereich lenken

- Taxonomien könne auch für nicht-funktionale Bereiche (→ Benutzbarkeit, Performanz, etc.) verwendet werden
- Taxonomie-Listen → versch. Veröffentlichungen, IEEE, Internet

3.4. Erfahrungsbasierte Verfahren

- Nutzung fachlicher Kompetenz & Intuition der Tester, Erfahrungen mit ähnlichen Anwendungen / Technologien
- Tests durchaus effektiv beim Auffinden von Fehlern, weniger gut bez. Bestimmung von Überdeckungsgraden oder für Wiederverwendbare Tests.
- Wenn Systemdoku schlecht, verfügbare Zeit knapp, Testteam ausgeprägtes Fachmissen über System hat → Erfahrungsbasiertes Testen = gute Alternative zu strukturierten Vorgehensweisen
- Erfahrungsbasiertes Testen schlecht: wenn Systeme detaillierte Testdoku / sehr gute Wiederholbarkeit benötigen, Testüberdeckungsgrad genau bewertbar sein muss
- Beim Einsatz dynamischer / heuristischer Ansätze → Einsatz von erfahrungsbasierten Tests. Dann Testen = eher Reaktion auf Ereignisse anstatt Verfolgung geplanter Vorgehensweise
- Gleichzeitige Testdurchführung & -auswertung
- Einige strukturierte Vorgehensweisen bei erfahrungsbasierten Tests = nicht durchweg dynamisch: Tests werden nicht vollkommen gleichzeitig entworfen und durchgeführt
- Erfahrungsbasiertes Testen liefert keine formalen Kriterien für Überdeckungsgrade

3.4.1. Intuitive Testfallermittlung (I Testfallermittlung, eng: Error Guessing)

- Einleitung: TA nutzen Erfahrung, um Fehler zu erraten, die ev. gemacht wurden, als Code erstellt wurde. Wenn die erwarteten Fehlerhandlungen identifiziert, bestimmt TA die Methoden zur Aufdeckung der daraus resultierenden Fehlerzustände. Bsp.: Wenn TA erwartet, dass SW Fehlerwirkungen zeigen wird bei ungültiger PW Eingabe → Tests zur Eingabe verschiedener Werte fürs PW Feld entwerfen. I auch bei Risikoanalyse nützlich
- Anwendbarkeit: Grunds. auf jeder Teststufe, haupts. Integrations- und Systemtest. Verfahren oft mit anderen Testverfahren im Einsatz, um Bandbreite der Testfälle zu erweitern. I Testfallermittlung auch effektiv beim Testen neuer SW Releases, um SW auf häufige Fehler und Fehlhandlungen zu testen, noch bevor gründlicheres & strukturbasiertes Testen erfolgt. Checklisten & Taxonomien können helfend unterstützen
- Einschränkungen/Schwierigkeiten: Überdeckung schwierig zu bestimmen, stark von Fachkompetenz & Erfahrung des TA abhängig. Am besten, wenn das Verfahren von erfahrenen Testern eingesetzt wird, der sich gut mit Fehlerarten auskennt, die häufig bei der Art des Codes vorkommen. I Testfallermittlung wird häufig eingesetzt aber selten dokumentiert → tiefe Reproduzierbarkeit
- Überdeckung: Mit Taxonomie durch zutreffende Datenfehler und Fehlerarten. Ohne Taxonomie: Überdeckung = f(Erfahrung & Wissen der Tester, verfügbare Zeit) Erfolg hängt stark davon ab wie der Tester Problembereiche trifft
- Fehlerarten: Durch Taxonomie vorgegeben, oder Fehlerzustände die der TA intuitiv errät und die durch spezifikationsorientierten Testverfahren ev. nicht gefunden worden wären

3.4.2. Checklisten-basiertes Testen (CL-basiertes Testen)

- Einleitung: Erfahrene Tas benutzen abstrakte, verallgemeinerte Listen mit Punkten die getestet werden müssen / nicht vergessen werden dürfen. Kann auch ein Satz von Vorschriften & Kriterien sein, gegen die das Produkt verifiziert werden muss. Diese Checklisten werden aus einer Reihe von Standards, Erfahrungen, sonstigen Überlegungen zusammengestellt. Bsp.: Checkliste mit Standards für GUI = Grundlage für Tests
- Anwendbarkeit: Auf jeder Teststufe, auch für Regressions- und Smoke-Tests. Am effektivsten mit erfahrenen Testern, welche die SW / Themenbereich der Checkliste gut kennt. Bsp.: Tester kennt sich mit GUIs generell gut aus, kennt aber die SW nicht. Da CL abstrakt → TA muss einzelne Lücken wie Testschritte füllen. Wartungsaufwand gering, CL für mehrere ähnliche Releases einsetzbar
- Einschränkungen/Schwierigkeiten: Reproduzierbarkeit tief da CL abstrakt gehalten → Untersch. Tester = untersch. Testergebnisse → breitere Überdeckung. CL können zu unbegründetem Vertrauen führen bez. erzieltem Überdeckungsgrad, da der Test von Einschätzung des Testers abhängt. CL können von detaillierten Testfällen / Listen abgeleitet werden und neigen dazu im Laufe der Zeit umfangreicher zu werden. CL-Wartung nötig → Sicherstellen, dass wichtige Aspekte der SW abgedeckt werden
- Überdeckung: Abhängig von der verwendeten CL. Da CL abstrakt, auch vom Tester abhängig
- Fehlerarten: Fehlerwirkungen, die als Folge variierender Daten oder in Zusammenhang mit der Ablaufsequenz oder dem allgemeinen Workflow auftreten. CL helfen das Testen aktuell zu halten, da auch neue Kombinationen von Daten & Prozessen während dem Testen erlaubt sind

3.4.3. Exploratives Testen (EX Testen)

- Einleitung: Tester lernen gleichzeitig SW und dessen Fehler kennen, planen vorgesehene Testaufgaben, entwerfen Tests, führen diese durch, berichten über Testergebnisse. Tester passen Testziele während Testdurchführung dynamisch, nur leichtgewichtige Doku
- Anwendbarkeit: Gutes EX Testen ist geplant, interaktiv und kreativ. Wenig Doku für SW nötig, daher wird EX Testen häufig angewendet, wenn Doku mangelhaft. EX Testen oft Erweiterung anderer Testverfahren, als Basis für zusätzliche Testfälle
- Einschränkungen/Schwierigkeiten: Management & Zeitplanung beim EX Testen schwierig. Überdeckung und Reproduzierbarkeit eingeschränkt. Managementmethode: Test-Chartas, welche die Aufgaben spezifizieren, die in einer Testsitzung abgedeckt werden sollen und den dafür benötigten Zeitrahmen eingrenzen. Am Ende der Testsitzung(en) → TM hält Abschlussbesprechung, um die Ergebnisse zu sammeln und Chartas für nächste Sitzung(en) zu bestimmen → Skalierbarkeit dieser Sitzungen für grössere Teams / Projekte schwierig. Verfolgbarkeit der Testsitzungen in Testmanagementsystem schwierig → EX Tests als Testfall-Artefakte im Testmanagementsystem ablegen → Zeit für EX Testen und geplante Überdeckung mit anderen Testaufgaben verfolgbar. Verbesserung der Reproduzierbarkeit = Verwendung von Capture/Playback-Werkzeugen → Root Cause Analysis zwar mühsam, aber alles dafür vorhanden
- Überdeckung: Test-Chartas erstellbar, die Aufgaben, Ziele und Arbeitsergebnisse spezifizieren. Bei explorativen Testsitzungen → spezifizierbar was erreicht werden soll.

In Test-Charta auch identifizierbar, worauf man sich konzentrieren sollte, was innerhalb & ausserhalb des Leistungsumfangs liegt, welche Ressourcen zur Durchführung der Tests vorzusehen sind. Testsitzung kann sich auf bestimmte Fehlerarten und andere potentielle Problembereiche fokussieren, die ohne Formalität von skriptbasierten Tests adressiert werden können

- Fehlerarten: Probleme mit Szenarien, die beim skriptbasierten funktionalen Test übersehen wurden, Probleme in funktionalen Grenzbereichen, Probleme in Zusammenhang mit dem Workflow, Performanz- und Sicherheitsprobleme

3.4.4. Anwendung des am besten geeigneten Testverfahrens

- Fehler- und erfahrungsbasierte Testverfahren nutzen Kenntnisse über Fehler und andere Erfahrungen im Testen, um Fehleraufdeckung zielgerichtet zu erhöhen.
- Reichen von Schnelltests (→ nicht formal) über geplante Testsitzungen bis hin zu Tests mit Testskripten (→ skriptbasiertes Testen)
- Besonders wertvoll, wenn:
  - Keine Spezifikationen vorhanden sind
  - Zu testendes System schlecht dokumentiert ist
  - Für Testentwurf und Erstellung der Testszenarien zu wenig Zeit vorhanden ist
  - Besonders erfahrene Tester im zu testenden Fachbereich &/ Technologie
  - Eine Erhöhung der Vielfalt gegenüber reinem skriptbasiertem Testen gesucht wird (→ höhere Überdeckung)
  - Betriebsausfälle analysiert werden sollen
- Fehler- und erfahrungsbasierte Testverfahren → nützlich, wenn mit spezifikationsorientierten Verfahren kombiniert
  → Lücken in Testüberdeckung schliessbar, welche aus systematischen Schwächen der Verfahren herrühren
- Es gibt nicht ein einziges perfektes Verfahren, welches für alle Situationen passt
- TA muss Vor- und Nachteile der einzelnen Verfahren kennen, um beste Verfahren / beste Kombination von Verfahren für bestimmte Situation auszuwählen
- Verschiedene Faktoren berücksichtigen, die die Auswahl beeinflussen können: Art des Projektes, Zeitpan, Zugang zu Informationen, Fachkompetenz der Tester, etc.)

4. Softwarequalitätsmerkmale

4.1. Einführung

- Inhalt dieses Kapitels: Wie wenden Tester Verfahren an, um die wichtigsten Qualitätsmerkmale für SW-Anwendungen und -Systeme zu bewerten?
- Beschreibung der Qualitätsmerkmale orientiert sich am ISO Standard 9126 → wird als Richtschnur verwendet
- Weitere nützliche Standards: z.B. ISO25000 → Ablösung von ISO9126 → ebenfalls nützlich
- ISO Qualitätsmerkmale unterteilt in Produktqualitätsmerkmale → Unterteilung in weitere Teileigenschaften möglich
- Folgende Tabelle zeigt Aufteilung in TA & TTA Lehrpläne:

Qualitätsmerkmal	Untermerkmale	Test Analyst	Techn. Test Analyst
Funktionalität	Richtigkeit, Angemessenheit, Interoperabilität, Einhaltung von Standards (Konformität)	X	
	Sicherheit		X
Zuverlässigkeit	SW-Reife (Robustheit), Fehlertoleranz, Wiederherstellbarkeit, Einhaltung von Standards (Konformität)		X
Benutzbarkeit	Verständlichkeit, Erlernbarkeit, Operabilität, Attraktivität, Einhaltung von Standards (Konformität)	X	
Effizienz	Performanz (Zeitverhalten), Ressourcennutzung, Einhaltung von Standards (Konformität)		X
Wartbarkeit	Analysierbarkeit, Änderbarkeit, Stabilität, Testbarkeit, Einhaltung von Standards (Konformität)		X
Portabilität	Anpassbarkeit, Installierbarkeit, Koexistenz, Austauschbarkeit, Einhaltung von Standards (Konformität)		X

- TA ist auch für Zugänglichkeitstests zuständig → Nicht in dieser Tabelle, dennoch Teil des Benutzbarkeitstests
- Einhaltung von Standards (Konformität) → bei allen Q-Merkmalen aufgeführt → v.a. für sicherheitskritische Systeme oder im regulatorischen Umfeld nötig. Bsp.: Einhaltung von Funktionalitätskonformität deutet darauf hin, dass Funktionalität einem bestimmten Standard entsprechen muss, z.B. ein bestimmtes Kommunikationsprotokoll für Datenaustausch mit einem Chip verwendet werden muss
- Standards variieren je nach Branche stark
- Für alle Qualitäts- und Teilmerkmale müssen typische Risiken erkannt werden, damit Teststrategie erstellbar
- Fürs Testen von Qualitätsmerkmalen besondere Beachtung schenken: Lebenszyklusabläufe, Werkzeuge, Verfügbarkeit der SW, Dokumentation, technisches Fachwissen
- Wenn keine geplante Vorgehensweise für jedes Merkmal & Testerfordernisse vorhanden →zu wenig Zeit für Planung, Vorbereitung und Durchführung
- Bei einigen Tests (z.B. Benutzbarkeit) müssen Spezialisten, umfangreiche Planung, spezielle Labors, bestimmte Werkzeuge, spezielle Testtätigkeiten und in den meisten Fällen erheblicher Zeitaufwand eingeplant werden, oft dafür auch Experten nötig
- Auch wenn TA nicht für alle Qualitätsmerkmale zuständig sind, müssen TA diese kennen → Überschneidungen
  Bsp. 1: Produkt, welches Performanztest nicht besteht, besteht oft auch den Benutzbarkeitstest nicht
  Bsp. 2: Produkt mit Problemen bei Interoperabilität einiger Komponenten → noch nicht bereit für Portabilitätstest

4.2. Qualitätsmerkmale bei fachlichen Tests

- Aufgabenschwerpunkt von TA = Funktionale Test → Was leistet das Produkt?
- Basieren i.A. auf Spezifikation / Anforderungsdokument, spez. Expertise in einem Fachgebiet, erwarteter Bedarf
- Funk. Tests unterscheiden sich je nach Teststufe & können vom SW-Lebenszyklus beeinflusst werden:
  - Integrationstest → Prüfen der Schnittstellenmodule für definierte Funktion
  - Systemtest → Prüfen der Funktionalität der gesamten Anwendung
  - Multisysteme → Funkionale End-to-End Tests
  - Agiles Umfeld → Beschränkung des funk. Testen auf Funktionen der aktuellen Iteration/Sprint → Regressionstest / Iteration = gesamte freigegebene Funktionalität
- Funktionale Qualitätsmerkmale, nachfolgende behandelt: Korrektheit & Angemessenheit & Interoperabilität
- Nicht-funktionale Tests, nachfolgend behandelt: Benutzbarkeit & Zugänglichkeit

4.2.1. Test der funktionalen Korrektheit

- Prüfung der Einhaltung von spezifizierten / impliziten Anforderungen an eine Anwendung, inkl. Richtigkeit von Berechnungen
- Viele Testverfahren möglich, siehe Kapitel 3
- Testorakel oft = Spezifikation / vorhandenes System
- In jeder Lebenszyklusphase durchführbar → Aufdeckung inkorrekter Handhabung von Daten / Situationen

4.2.2. Angemessenheitstest

- Bewerten und validieren, ob die Menge von Funktionen für die vorgesehenen spezifizierten Aufgaben eignen
- Tests können auf Anwendungsfällen (use cases) basieren
- Werden meist beim Systemtest durchgeführt, oder in späteren Stufen des Integrationstests
- Fehlerzustände = Hinweis darauf, dass System Erfordernisse der Benutzer nicht in akzeptabler Weise erfüllen wird

4.2.3. Interoperabilitätstests

- Prüfen Informationsaustausch >2 Systeme / Komponenten
- Tests müssen alle vorgesehenen Umgebungen abdecken (inkl. Varianten von HW, SW, Middleware, Betriebssystemen) um sicherzustellen, dass Datenaustausch korrekt funktioniert
- In Praxis → möglicherweise nur für relativ kleine Anzahl von Umgebungen machbar → Begrenzung von repräsentativen Umgebungen
- Für Spezifikationen der Interoperabilitätstests → Kombinationen der vorgesehenen Zielumgebungen identifizieren, konfigurieren & Testteam zur Verfügung stellen

- Diese Umgebungen dann mit ausgewählten funktionalen Testfällen testen, welche die versch. Datenaustauschstellen prüfen
- SW mit guten Interoperabilitätseigenschaften → leichte Integration mit anderen Systemen, ohne grössere Anpassungen
- Messung von Interoperabilität → # notwendiger Änderungen, inkl. dessen Aufwand
- Designmerkmale, die im Fokus stehen können:
    - Verwendung von industrieüblichen Kommunikationsstandards (z.B. XLM)
    - Fähigkeit der SW, Kommunikationsanforderungen anderer Systeme automatisch zu erkennen
- Interoperabilitätstests besonders wichtig für kommerzielle Standard-SW & -Werkzeuge / Multisysteme
- Während Komponentenintegrations- & Systemtests → Fokus auf Zusammenwirken des Systems mit Umgebung
- Während Systemintegrationstest → Fokus auf Zusammenwirken des Systems mit anderen Systemen. Da Systeme auf mehreren Ebenen interoperieren, muss TA diese Interaktionen verstehen & Bedingungen dafür schaffen
  Bsp.: 2 Systeme tauschen Daten aus → TA muss Daten und Transaktionen für diesen Datenaustausch erzeugen
- Zu beachten: Oft nicht alle Interaktionen in den Anforderungen klar spezifiziert, oft nur in Doku für Systemarchitektur & Systementwurf → TA muss diese Doku untersuchen, damit Punkte für Datenaustausch testbar
- Testverfahren wie ET, Zustandsübergangsdiagramme, Anwendungsfälle und kombinatorisches Testen → Alle im Interoperabilitätstest anwendbar
- Typische Fehlerzustände = inkorrekter Datenaustausch zwischen interagierenden Komponenten

4.2.4. Benutzbarkeitstests

- Mögliche Personengruppen: Von Kindern / Menschen mit besonderen Einschränkungen bis IT-Experten
- Einige nationale Verbände (→ British Royal National Institute for the Blind) empfehlen Webseiten für Behinderte, Blinde, Sehbehinderte oder Taube nutzbar zu machen
- Prüfung, ob Webseite auch für diese Personengruppen nutzbar ist, verbessert u.U. auch Benutzbarkeit aller anderen
- Benutzbarkeitstests prüfen: Wie einfach ist es das System zu nutzen / erlernen?
- Folgendes wird dabei gemessen:
    - Effektivität: Eignung der SW für Nutzer, spezifizierte Ziele in spez. Anwendungskontext zu erreichen
    - Effizienz: Eignung der SW für Nutzer, der Effektivität angemessene Ressourcen einzusetzen
    - Zufriedenheit: Eignung der SW, die Nutzer in spez. Anwendungskontext zufrieden zu stellen
- Messbare Merkmale:
    - Verständlichkeit: SW-Merkmale welche Nutzeraufwand beeinflussen, um logisches Konzept zu erkennen
    - Erlernbarkeit: SW-Merkmale welche Nutzeraufwand beeinflussen, um Anwendung zu erlernen
    - Operabilität: SW-Merkmale welche Nutzeraufwand beeinflussen, um Aufgaben effektiv & effizient auszuführen

  - Attraktivität: Eigenschaft der SW, dass sie dem Nutzer gefällt
- Benutzbarkeitstests werden i.d.R. in 2 Stufen durchgeführt:
  - Formativer Benutzbarkeitstest → Iterativer Test während Entwurfs- und Prototyping-Stufen → Identifikation von Entwurfsfehlern, die für Benutzbarkeit relevant sind → Hilft den Entwurf zu «formen»
  - Summativer Benutzbarkeitstest → Werden nach Realisierung durchgeführt, um Benutzbarkeit zu messen → Identifikation von Problemen mit fertigen Komponente / System
- Tester soll Wissen / Expertise in folgenden Wissensbereichen haben: Soziologie, Psychologie, Einhaltung nationaler Standards / Vorschriften (inkl. Zugänglichkeitsstandards), Ergonomie

4.2.5. Benutzbarkeitstests durchführen

- Implementierte System soll unter möglichst realitätsnahen Bedingungen validiert werden → Benutzbarkeitslabor mit Videokameras, nachgebautes Büro, Review-Gruppen, Nutzer, etc. → Beobachtung der Wirkung des tatsächlichen Systems auf echte Personen durch Entwickler
- Formale Benutzbarkeitstests → mit echten Benutzern / -vertreter durchführen, die oft für ihre Aufgabe vorbereitet werden müssen
- In freien Tests → Experimentation der Benutzer mit SW, damit Beobachter bestimmen können, wie einfach / schwierig es für Benutzer ist herauszufinden, wie sie ihre Aufgaben erfüllen können
- Viele Benutzbarkeitstests können vom TA als Teil anderer Tests durchgeführt werden, z.B. beim funkt. Systemtest
- Für Konsistenz → Ergonomierichtlinien hilfreich → ohne: schwierig zu bestimmen was «unakzeptable» Benutzbarkeit ist
- Bsp.: Ist es unzumutbar, wenn ein Benutzer 10 Mausklicks benötigt, um sich einzuloggen? → Ohne Richtlinien schwierig für TA zu entscheiden ob ok oder nicht
- Wichtig, dass verifizierbare Benutzbarkeitsspezifikationen in den Anforderungen enthalten sind & dass ein Satz von Ergonomierichtlinien existiert die für alle ähnlichen Projekte gelten
- Folgende Punkte sind durch diese RiLi abzudecken: Zugänglichkeit von Anleitungen, Klarheit von Eingabeaufforderungen, Anzahl der Klicks für die Durchführung der Aufgabe, Fehlermeldungen, Verarbeitungsindikatoren (→ zeigt dem Benutzer an, dass das System momentan in der Verarbeitung beschäftigt ist & keine weiteren Eingaben annehmen kann), RiLis über Bildschirmlayout, Verwendung von Farben und Geräuschen und weitere Faktoren, die Einfluss auf Benutzererfahrung haben

4.2.6. Benutzbarkeitstestspezifikation

- Die wichtigsten Verfahren für Benutzbarkeitstests:
  - Inspektionen, Bewertungen oder Reviews
  - Dynamische Interaktion mit Prototypen
  - Verifizierung und Validierung der tatsächlichen Implementierung
  - Durchführung von Befragungen

Inspektionen, Bewertungen oder Reviews

- Inspektionen / Reviews von Anforderungsspezifikationen & Entwürfen bez. Benutzbarkeit → Erhöhung der Nutzerbeteiligung & kosteneffektiv, weil sie Probleme früh erkennen
- Mit heuristischer Evaluation (→ system. Inspektion des Entwurfs einer Benutzerschnittstelle auf Benutzbarkeit) lassen sich Probleme der Benutzbarkeit im Entwurf aufdecken, sodass diese im iterativen Entwurfsprozess bearbeitet werden können → Kleines Gutachterteam prüft die Schnittstelle und ihre Konformität mit anerkannten Grundsätzen der Benutzbarkeit (→ heuristische Grundsätze der Ergonomie)
- Reviews sind effektiver, wenn die Benutzerschnittstelle sichtbar ist
  Bsp.: Screenshots sind leichter zu verstehen & zu interpretieren als Beschreibung einer bestimmten Bildschirmfunktionalität in Textform
- Visualisierung wichtig, damit ein angemessenes Review der Dokumentation auf Benutzbarkeit möglich ist

Dynamische Interaktion mit Prototypen

- Wenn Prototypen entwickelt werden, sollte TA sich mit diesen beschäftigen und den Entwicklern bei der Weiterentwicklung behilflich sein, indem sie Benutzerfeedback in Entwurf einfliessen lassen
- So lassen sich Prototypen verfeinern & Nutzer bekommt realistischen Einblick zu Look and Feel des fertigen Produktes und wie es zu benutzen sein wird

Verifizierung und Validierung der tatsächlichen Implementierung

- Falls Benutzbarkeitsmerkmale der SW in Anforderungen spezifiziert (z.B. Anzahl Mausklicks für bestimmte Aufgabe), dann sollten Testfälle entworfen werden, die verifizieren, dass die spezifizierten Eigenschaften in Implementierung berücksichtigt wurden
- Zur Validierung der tatsächlichen Implementierung können Benutzbarkeitstestszenarien aus Tests entwickelt werden, die für den funktionalen Systemtest spezifiziert wurden. Dabei werden statt der funktionalen Ergebnisse die spezifischen Benutzbarkeitsmerkmale gemessen, z.B. Erlernbarkeit & Operabilität
- Spezifischen Benutzbarkeitstestszenarien für das Testen von Syntax und Semantik sind entwickelbar
- Syntax = Aufbau / Grammatik der Schnittstelle (z.B. was kann in Eingabefeld eingegeben werden?)
- Semantik = Bedeutung und Zweck der Schnittstelle (z.B. sinnvolle & aussagekräftige Systemmeldungen und -ausgaben an Nutzer
- Black-Box-Verfahren (→ siehe Abschnitt 3.2), insbesondere Anwendungsfälle, die entweder in Textform oder in UML (Unified Modeling Language) definiert sind, werden manchmal im Benutzbarkeitstest angewendet
- Testszenarien für Benutzbarkeitstests sollten Anleitungen für Nutzer enthalten, Zeiträume für Interviews vor & nach den Tests vorsehen, um Nutzer anzuweisen bzw. um von ihnen Rückmeldungen einzusammeln, sowie ein vereinbartes Protokoll für die Durchführung der Testsitzungen haben
- Das Protokoll beschreibt wie der Test ausgeführt wird, den zeitlichen Ablauf, Protokollierung und Aufzeichnung der Testsitzung sowie die vorgesehenen Methoden für Befragung und Beaufsichtigung

Durchführung von Umfragen und Fragebögen

- Damit lassen sich Erkenntnisse & Feedback über das Verhalten der Anwender bei Systemnutzung sammeln
- Standardisierte & öffentlich zugängliche Umfragen wie etwa SUMI (SW Usability Measurement Inventory) und WAMMI (Website Analysis & MeasurMment Inventory) → ermöglichen Benachmarking gegen eine DB mit früheren Benutzbarkeitsmessungen
- SUMI liefert auch konkrete Benutzbarkeitsmasse, die sich als Testende- oder Abnahmekriterien verwenden lassen

4.2.7. Zugänglichkeitstests

- Wichtig, SW für Menschen mit besonderen Bedürfnissen / Einschränkungen (inkl. Behinderte) zu prüfen
- Ähnlich wie Benutzbarkeit, muss auch Zugänglichkeit in Entwurfsphase berücksichtigt werden
- Erfolgen meist in Integrationsstufen, über Systemtest bis hin zu Abnahmetest
- Fehler ergeben sich aus Tatsache, dass die SW die spez. Vorschriften / Standards nicht erfüllt

5. Reviews

5.1. Einführung

- Erfolgreiches Review = Planung & Durchführung & Nachbereitung
- TA sind aktiv am Review-Prozess dabei und bringen ihre Perspektive ein
- Sie sollten ein formales Review-Training erhalten haben, sodass sie ihre jeweilige Rolle bei Review besser verstehen
- Alle Reviewteilnehmer müssen vom Nutzen guter Review überzeugt sein
- Reviews leisten den grössten einzelnen & kosteneffektivsten Beitrag zur Qualität
- Unabhängig von der Reviewart, muss TA genug Zeit für Vorbereitung einplanen um zu prüfen: Arbeitsergebnis, referenzierte Dokumente, Konsistenz von Arbeitsergebnis mit diesen Dokumenten, Fehlendes im Arbeitsergebnis
- Zu gutem Review gehört: Inhalte verstehen, zu bestimmen ob etwas fehlt und wenn ja was, verifizieren, dass das beschriebene Produkt mit anderen bereits / derzeit entwickelten Produkten konsistent ist
  Bsp.: Review eines Stufenkonzepts für Integrationstest → TA muss auch Objekte berücksichtigen, die integriert werden sollen: Welche Bedingungen müssen dazu erfüllt sein? Abhängigkeiten, die dokumentiert werden sollen? Daten verfügbar, um Integrationsstellen zu testen? → Review berücksichtigt auch die Interaktion des Review-Gegenstandes mit anderen Objekten des Systems
- Autor eines Reviewdokumentes kann sich kritisiert fühlen → Kommentare sollten immer konstruktiv formuliert sein und am Reviewgegenstand orientieren, nicht am Autor
  Bsp. Schlecht: «Die Anforderung ist zweideutig, die wird keiner verstehen können»
  Bsp. Gut: «Ich verstehe nicht, was ich testen soll um zu verifizieren, ob diese Anforderung korrekt implementiert wurde. Können sie mir helfen, das besser zu verstehen?»
- Aufgabe des TA im Review: Sicherstellen, dass die im Arbeitsergebnis gelieferte Info ausreichend sind um das Testen zu unterstützen

- Wenn Info nicht vorhanden, nicht klar und eindeutig, nicht detailliert genug → Fehler, vom Autor zu korrigieren
- Positive anstatt kritische Grundhaltung trägt dazu bei, dass Kommentare besser angenommen werden und Sitzung produktiver wird

5.2. Checklisten in Reviews verwenden

- Checklisten helfen mit, beim Review bestimmte Punkte im Reviewverlauf zu prüfen
- Helfen mit, Review zu «entpersonalisieren»
- Checklisten können allgemein gehalten sein und für alle Reviews gültig sein oder spezifisch für bestimmte Qualitätsmerkmale, Themenbereiche oder Dokumente
  Bsp.: Allgemeine Checkliste → Allgemeine Eigenschaften des Dokuments prüfen wie z.B. eindeutige Kennung, keine Verweise vorhanden auf Punkte die noch offen sind, Formatierung und ähnliche Elemente korrekt / konform sind. Spezifische Checkliste für Anforderungsdokument → Begriffe wie «soll» / «sollte» richtig verwenden, jede Anforderung testbar? Auch Format der Anforderungen kann darauf hinweisen, welche Checkistenart zu verwenden ist. Für Anforderungsdokument in Textform gelten andere Review-Kriterien als für eines beruhend auf Diagrammen
- Checklisten können auch auf spezifische Kompetenzen der Programmiere / Systemarchitekten oder Tester ausgerichtet sein
- Checkliste, die für Anforderungen, Anwendungsfälle oder User Stories verwendet werden, haben anderen Fokus als Checklisten, die für Programmcode oder Systemarchitektur verwendet werden
- Punkte (nur Beispiele!) für Checkliste für Anforderungen:
  - Testbarkeit
  - Abnahmekriterien
  - Verfügbarkeit einer Anwendungsfall-Aufrufstruktur, falls zutreffend
  - Eindeutige Identifikation bzw. Kennung jeder Anforderung / Anwendungsfall / User Story
  - Versionierung der einzelnen Anforderungen / Anwendungsfälle / User Stories
  - Rückverfolgbarkeit jeder einzelnen Anforderung von den Anforderungen des Fachbereichs / Marketings
  - Verfolgbarkeit zwischen Anforderungen und Anwendungsfällen
- Wenn Anforderung nicht testbar, liegt ein Fehlerzustand in der Anforderung vor. Nicht testbar bedeutet, dass die Anforderung so spezifiziert ist, dass der TA nicht bestimmen kann, wie sie getestet werden soll
  Bsp.: «Die SW soll sehr benutzerfreundlich sein» → nicht testbar. Besser: «Die SW muss den Benutzbarkeitsstandards entsprechen, die im Benutzbarkeitsstandard-Dokument spezifiziert sind» → testbar → es handelt sich übrigens um eine übergreifende Anforderung, die jedes Element der Benutzerschnittstelle betrifft → kann viele einzelne Testfälle hervorbringen. Zudem ist Verfolgbarkeit von dieser Anforderung bzw. vom Benutzbarkeitsstandard-Dokument zu den Testfällen kritisch: Gibt es Änderungen der Benutzbarkeitsspezifikation, auf welche in der Anforderung verwiesen wurde, müssen alle Testfälle geprüft und ggf. aktualisiert werden
- Anforderung auch dann untestbar, wenn Tester nicht bestimmen kann, ob Test bestanden wurde oder nicht, oder wenn kein Test entwickelt werden kann, der bestanden werden kann
  Bsp.: System soll 100% der Zeit zur Verfügung stehen, 24h * 365/266 Tage im Jahr
- Punkte für Checkliste für Anwendungsfälle:

    - Ist der Hauptpfad (Szenario) genau spezifiziert?
    - Sind alle alternativen Pfade (Szenarien) identifiziert, einschliesslich der Fehlerbehandlung?
    - Sind die Meldungen der Benutzerschnittstelle spezifiziert?
    - Gibt es nur einen Hauptpfad? → Sollte nur einen geben! Sonst: mehrere Anwendungsfälle nötig!
    - Ist jeder Pfad testbar?
- Punkte für einfache Checkliste für Benutzbarkeits-Reviews:
    - Ist jedes Feld und dessen Funktion spezifiziert?
    - Sind alle Fehlermeldungen spezifiziert?
    - Sind alle Benutzeraufforderungen spezifiziert und konsistent?
    - Ist die Tabfolge der Felder spezifiziert?
    - Gibt es alternativ zu Mauskationen auch Keyboardeingaben?
    - Sind Tastenkombinationen («Shortcuts») für den Benutzer spezifiziert?
    - Gibt es Abhängigkeiten zwischen den Feldern (z.B. Datum muss später sein als ein anderes Datum)
    - Ist ein Bildschirmlayout vorgegeben?
    - Entspricht das Bildschirmlayout den spezifizierten Anforderungen?
    - Gibt es eine Anzeige für den Benutzer, die zeigt, dass das System gerade verarbeitet?
    - Erfüllt der Bildschirm die Mindestanzahl von Mausklicks (falls spezifiziert)?
    - Ist Navigationsfluss für Benutzer logisch und basierend auf den Anwendungsfall?
    - Erfüllt Bildschirm etwaige Anforderungen bezüglich Erlernbarkeit?
    - Gibt es Hilfetexte für den Benutzer?
    - Gibt es schwebende Textanzeigen?
    - Wird dies für den Benutzer «attraktiv» sein (→ subjektive Bewertung)?
    - Sind die verwendeten Farben konsistent mit anderen Anwendungen und mit Unternehmensstandards?
    - Werden die Soundeffekte richtig verwendet und sind sie konfigurierbar?
    - Erfüllt der Bildschirm die Anforderungen bezüglich Lokalisierung?
    - Ist für den Benutzer ersichtlich, was zu tun ist (Verständlichkeit → subjektive Bewertung)?
    - Kann sich der Benutzer daran erinnern, was zu tun ist (Erlernbarkeit → subjektive Bewertung)?
- Agile Projekte: Anforderungen = User Stories → kleine Inkremente demonstrierbarer Funktionalitäten. Wenn sich ein Anwendungsfall um eine Benutzertransaktion handelt, die mehrere Bereiche von Funktionalität durchläuft, ist eine User Story eine isolierte Einheit, deren Umfang im Allgemeinen durch die Zeit bestimmt wird, die für ihre Entwicklung notwendig ist
- Punkte für Checkliste für User Stories:
    - Ist die Story angemessen für die vorgesehene Iteration / Sprint?
    - Sind die Abnahmekriterien spezifiziert und sind diese testbar?
    - Ist die Funktionalität genau definiert?
    - Gibt es Abhängigkeiten zwischen dieser User Story und anderen User Stories?
    - Ist die User Story priorisiert?
    - Umfasst die User Story ein Inkrement von Funktionalität?
- Falls User Story eine neue Schnittstelle definiert, dann ist es besser, eine allgemeine Checkliste (siehe oben) und eine detaillierte Benutzerschnittstellen-Checkliste zu verwenden
- Checklisten können an folgende Grundlagen angepasst werden:

  - Unternehmen / Organisation (z.B. Berücksichtigung der Unternehmenspolitik, -standards, -konventionen)
  - Projekt / Entwicklungsaufwand (z.B. Schwerpunkt, technische Standards, Risiken)
  - Review-Objekt (→ Code-Reviews an Programmiersprache anpassen)
- Gute Checklisten decken Probleme auf und sie initiieren Diskussionen über andere Punkte, auf die die Checkliste möglicherweise nicht verweist
- Wenn bei Review verschiedene Checklisten kombiniert werden → probates Mittel, dass das Review die beste Qualität für das Arbeitsergebnis erzielt
- Durch Verwendung von Standard-Checklisten sowie durch Entwicklung eigener, unternehmensinterner Checklisten, kann der TA einen effektiven Beitrag bei Reviews leisten

6. Fehlermanagement

6.1. Einleitung

- TA untersuchen das Systemverhalten aus geschäftlichen und Benutzerperspektive. Bsp.: Würden Nutzer wissen, was zu tun ist, wenn diese Meldung erscheint / wenn das System sich so verhält?
- TA bestimmt, ob sich das System korrekt verhält, indem er tatsächliche und erwartete Ergebnisse vergleicht
- Eine Anomalie (=Abweichung) = unerwartetes Ereignis, das näher untersucht werden muss
- Eine Anomalie kann eine Fehlerwirkung sein, die durch einen Fehlerzustand verursacht wird
- Eine Anomalie muss kein Fehlerzustand sein und kann zur Erstellung eines Fehlerberichts führen oder nicht
- Fehlerzustand = tatsächliches Problem, das festgestellt wurde und behoben werden muss

6.2. Wann lässt sich ein Fehlerzustand aufdecken?

- Ein Fehlerzustand wird durch statische Tests gefunden
- Die Symptome eines Fehlerzustands = Fehlerwirkung → wird durch dynamische Tests aufgedeckt
- Für jede Phase des SW-Lebenszyklus sollte es Methoden zum Erkennen & Beseitigen von Fehlerwirkungen geben
- Entwicklungsphase → Code-Reviews & Design-Reviews zum Aufdecken von Fehlerzuständen
- Dynamische Tests → Testfälle fürs Finden von Fehlerwirkungen
- Je früher ein Fehlerzustand entdeckt und korrigiert wird, desto geringer die Kosten für Qualität des Gesamtsystems
- Statische Tests können Fehlerzustände aufdecken, bevor dynamisches Testen möglich ist → einer der Gründe, weshalb statische Tests kosteneffektive Methode für Entwicklung qualitativ hochwertiger SW ist

- Mit Hilfe des Fehlerverfolgungssystems sollte es möglich sein die Lebenszyklusphase aufzuzeichnen für: Wann ist der Fehlerzustand entstanden? Wann ist der Fehlerzustand aufgedeckt worden? → Falls beide Phasen identisch → Fehlereindämmung innerhalb der Phase gelungen. Bsp.: Inkorrekte Anforderung, welche in Review gefunden und gleich korrigiert wird → effizienter Einsatz eines Reviews, verhindert, dass der Fehlerzustand später zusätzlichen Aufwand verursacht und so teurer wäre
- Wenn inkorrekte Anforderung übersehen wird («entkommt»), und dann vom Entwickler implementiert & vom TA getestet und dann erst vom Benutzer im Abnahmetest gefunden wird, ist die ganze Arbeit an dieser Anforderung Zeitverschwendung & kann zum Vertrauensverlust des Nutzers führen

6.3. Die Pflichtfelder in Fehlerberichten

- Pflichtfelder sollen ausreichend Info liefern, damit entsprechende Massnahmen zur Fehlerbehebung ergriffen werden können. Ein für die Feststellung der Massnahmen geeigneter Fehlerbericht ist:
  - Vollständig – der Bericht enthält alle benötigte Infos
  - Kurz und prägnant – der Bericht enthält keine unnötigen & irrelevanten Infos
  - Richtig – die Infos im Bericht sind richtig, Erwartungswert & richtiger Wert sind genau spezifiziert, Repro-Schritte vorhanden
  - Objektiv – der Bericht ist professionell verfasst und bezieht sich auf die Tatsachen
- Fehlerbericht sollte in Datenfelder unterteilt sein → Je besser diese Felder definiert sind, desto leichter ist es den Fehler zu berichten sowie Berichte über Fehlertrends und sonstige zusammenfassende Berichte zu erstellen
- Wenn für ein Feld mehrere Optionen möglich sind → Drop-Down-Listen verwenden → nur dann effektiv, wenn diese nicht zu lang ist
- Unterschiedliche Fehlerarten = unterschiedliche Fehlerberichtsinformationen nötig → Fehlermanagementwerkzeug soll flexibel sein und je nach Fehlerart die richtigen Felder abfragen
- Daten sollen in spezifischen Feldern aufgezeichnet werden, die im Idealfall mit Datenvalidierungsfunktion ausgestattet sind, um Fehler bei Dateneingabe zu vermeiden und um effektive Berichterstattung sicherzustellen
- Fehlerberichte gibt es für aufgedeckte Fehlerwirkungen während funktionalen & nicht-funktionalen Tests
- Info im Fehlerbericht sollte zum Ziel haben, das Szenario wo das Problem entdeckt wurde, genau zu identifizieren, inkl. der benötigten Schritte und Daten, um das Szenario zu reproduzieren, inkl. Erwartungswert & Messwert
- Nicht-funktionale Fehlerberichte benötigen mehr Details über Umgebung, Performanz-Parameter (z.B. Last), Reihenfolge der Schritte und Erwartungswert
- In Fehlerbericht über Benutzbarkeitsfehlerzustand ist es wichtig zu beschreiben, welche SW-Reaktion der Nutzer erwartet hat
  Bsp.: Benutzbarkeits-Standard legt fest, dass eine Aktion <4 Mausklicks haben darf → im Fehlerbericht soll stehen, wie viele Mausklicks tatsächlich benötigt wurden im Gegensatz zum erwähnten Standard
- Falls kein Standard vorhanden und die nicht-funktionalen Qualitätsmerkmale der SW in den Anforderungen fehlen, kann Tester einen Test mit «vernünftiger Person» durchführen um zu bestimmen, ob Benutzbarkeit akzeptabel ist → Erwartungen dieser Testperson muss im Fehlerbericht klar dargelegt werden

- Da die nicht-funktionalen Anforderungen in Anforderungsdokumentation oft fehlen, ist das Dokumentieren des erwarteten und tatsächlichen Systemverhaltens in Zusammenhang mit nicht-funktionalen Fehlerwirkungen für Tester oft besondere Herausforderung
- Fehlerberichte haben normalerweise zum Ziel, die Behebung des Problems herbeizuführen. Die Infos im Fehlerbericht müssen auch die richtige Fehlerklassifizierung enthalten sowie Risikoanalyse & Prozessverbesserungen unterstützen

6.4. Fehlerklassifizierung

- Fehlerbericht kann während seines Lebenszyklus verschiedene Klassifizierungsstufen haben
- Richtige Klassifizierung von Fehlern = integraler Bestandteil einer korrekten Fehlerberichterstattung
- Klassifizierung wird genutzt, um Fehlerzustände zu gruppieren, um die Effektivität des Testens und des Entwicklungslebenszyklus zu bewerten & um interessante Trends zu erkennen
- Folgende Informationen sind häufig Bestandteil neuer Fehlerzustände:
    - Projektaktivität, die durchgeführt wurde, als der Fehlerzustand entdeckt wurde (→ Review, Audit, Inspektion, Programmieren, Testen)
    - Projektphase, in welcher der Fehlerzustand entstanden ist (falls bekannt) (z.B. Anforderungen, Entwurf, Feinentwurf, Implementierung)
    - Projektphase, in der der Fehlerzustand entdeckt wurde (z.B. Anforderungen, Entwurf, Feinentwurf, Implementierung, Code-Review, Modultest, Integrationstest, Systemtest, Abnahmetest)
    - Vermutete Ursache des Fehlerzustands (z.B. Anforderungen, Entwurf, Schnittstelle, Programmcode, Daten)
    - Wiederholbarkeit (einmal, sporadisch, reproduzierbar)
    - Symptome (Systemabsturz, Systemstillstand, Benutzerschnittstellenfehler, Systemfehler, Performanz)
- Nach Analyse des Fehlerzustands, kann weitere Klassifizierung erfolgen:
    - Grundursache – der Fehler der gemacht wurde, und aus dem der Fehlerzustand resultierte (z.B. Prozess, Programmierfehler, Benutzerfehler, Testfehler, Konfigurationsproblem, Datenproblem, Fremdsoftware, externes SW-Problem, Dokumentationsproblem)
    - Quelle – das Arbeitsprodukt, in dem der Fehler gemacht wurde (z.B. Anforderungen, Entwurf, Feinentwurf, Architektur, Datenbankdesign, Benutzerdokumentation, Testdokumentation)
    - Art – (z.B. Logikproblem, Berechnungsproblem, zeitliches Problem, Datenhandling, Verbesserung)
- Nach Behebung des Fehlers / Zurückstellung sind noch weitere Infos zum Fehlerzustand verfügbar:
    - Lösung (z.B. Programmänderung, Dokumentationsänderung, zurückgestellt, kein Problem, Duplikat)
    - Korrekturmassnahme(n) (z.B. Anforderungs-Review, Code-Review, Modultest, Konfigurationsdokumentation, Datenvorbereitung, keine Änderung erfolgt)
- Zusätzliche Klassifizierungen → Schweregrad & Priorität

- Zudem sinnvoll (je nach Projektkontext) → Auswirkungen auf: Sicherheit, Projektplan, Projektkosten, Projektrisiken, Projektqualität → wichtig, wenn Vereinbarung gefunden werden soll, wie schnell der Fehler zu beheben ist
- Gruppierung & Klassifizierung bei Schliessung des Fehlers: behoben/verifiziert, geschlossen/kein Problem, zurückgestellt, offen/nicht behoben → Lebenszykluszustände der Fehlerzustände
- Konsistente Verwendung der Klassifizierungsdaten nötig, damit diese nützlich sind
- Bei zu vielen Klassifizierungswerten ist das Bearbeiten von Fehlern zeitintensiv → Anpassen bis Ausgewogenheit!

6.5. Grundursachenanalyse

- Grund der Grundursachenanalyse = zu bestimmen, was den Fehlerzustand verursacht hat & Daten zu liefern zur Unterstützung von Prozessänderungen
- Diese Prozessänderungen sollen Grundursachen eliminieren, welche für einen bedeutenden Teil der Fehlerzustände verantwortlich sind
- Grundursachenanalyse wird meist von Person durchgeführt, die einen Fehlerzustand untersucht und entweder das Problem behebt oder bestimmt, dass das Problem nicht behoben werden kann oder nicht behoben werden soll
- In der Regel ist diese Person der Entwickler
- Vorläufige Grundursachenanalyse erfolgt häufig durch TA, der auf Basis von Erfahrung eine Vermutung über die wahrscheinliche Ursache des Problems äussert.
- Nach Bestätigung der Behebung des Problems, verifiziert der TA die vom Entwickler eingetragene Grundursache
- In Zusammenhang mit Bestimmung der Grundursache wird häufig auch die Phase bestimmt / bestätigt, in der der Fehlerzustand entstanden ist
- Typische Grundursachen von Fehlerzuständen sind:
    - Anforderung: Unklar / Fehlend / Falsch
    - Inkorrekte Implementierung: des Entwurfs / der Schnittstelle
    - Logikfehler im Code
    - Berechnungsfehler
    - HW Fehler
    - Schnittstellenfehler
    - Ungültige Daten
- Die Infos über Grundursachen werden zusammengeführt, um Themen zu bestimmen, die zu Fehlerzuständen führen
  Bsp. 1: Viele Fehlerzustände durch unklare Anforderungen → mehr Aufwand in Anforderungs-Reviews investieren
  Bsp. 2: Mehrere Entwicklerteams, Probleme bei Implementierung von Schnittstellen → gemeinsame Entwurfssitzungen
- Wenn Infos zu Grundursachen für Prozessverbesserungen genutzt werden, kann ein Unternehmen den Nutzen effektiver Prozessänderungen überwachen, denn damit lassen sich die Kosten von Fehlerzuständen quantifizieren, die auf eine bestimmte Grundursache zurückzuführen sind
- Dies kann hilfreich sein, wenn es darum geht, finanzielle Mittel für Prozessveränderungen zu bekommen, für die zusätzliche Werkzeuge & Ausrüstung beschafft werden müssen oder die sich auf den Zeitplan auswirken

7. Testwerkzeuge

7.1. Einführung

- Testwerkzeuge → Effizienz & Effektivität des Testens in Bezug auf Testaufwand steigt → aber nur dann, wenn die Testwerkzeuge fachgerecht eingesetzt werden
- Testwerkzeuge müssen gemanagt werden
- Es bestehen grosse Unterschiede in Ausgereiftheit und Anwendbarkeit der Werkzeuge, der Markt ändert sich stets
- Meist von kommerziellen Anbietern sowie von verschiedenen Freeware- oder Shareware-Webseiten

7.2. Testwerkzeuge und Automatisierung

- Grossteil der Arbeit eines TA = effizienter Einsatz von Testwerkzeugen
- TA muss wissen, welche Werkzeuge wann einzusetzen sind
- Dies kann Effizienz des TA erhöhen und dazu beitragen, in verfügbaren Zeit bessere Testabdeckung zu erzielen

7.2.1. Testentwurfswerkzeuge

- Werden verwendet, um die Erzeugung von Testfällen und Testdaten für das Testen zu unterstützen
- Werkzeuge verwenden bestimmte Formate der Anforderungsspezifikation (z.B. UML) oder Eingaben des TA
- Testentwurfswerkzeuge häufig so entworfen & gebaut, damit sie mit bestimmten Formaten & Produkten zusammenarbeiten können, wie z.B. spezifische Anforderungsmanagementwerkzeuge
- Testentwurfswerkzeuge liefern dem TA Informationen darüber, welche Tests er braucht, um die angestrebte Testabdeckung, Vertrauen ins System oder die Massnahmen zur Reproduzierung des Produktrisikos zu erzielen
  Bsp.: Klassifikationsbaumwerkzeuge, die eine Menge von Kombinationen generieren und darstellen, die für eine Erfüllung vordefinierter Überdeckungskriterien erforderlich sind → TA kann dann auszuführende Testfälle festlegen

7.2.2. Testdateneditoren und -generatoren

- Einige dieser Werkzeuge sind in der Lage, ein Dokument (z.B. Anforderungsdokument) oder sogar Quellcode zu analysieren und die Daten zu bestimmen, die beim Testen für einen angestrebten Überdeckungsgrad benötigt werden
- Andere dieser Werkzeuge können einen Datensatz aus einem Produktivsystem so bereinigen / anonymisieren, dass die darin enthaltenen persönlichen Daten entfernt werden, die interne Integrität der Daten jedoch erhalten bleibt. Diese bereinigten Daten können dann für Testzwecke verwendet werden, ohne eine Sicherheitslücke oder Missbrauch der persönlichen Daten zu riskieren

- Vor allem dann von Bedeutung, wenn grosse Mengen realistischer Daten benötigt werden
- Andere Testdatengeneratoren können Testdaten aus einer vorhandenen Menge von Eingabeparametern generieren (z.B. für Tests mit Zufallsdaten)
- Manche dieser Werkzeuge können die DB-Struktur analysieren um zu bestimmen, welche Eingaben vom TA benötigt werden

7.2.3. Automatisierte Testausführungswerkzeuge

- Werden überwiegend von TA auf allen Teststufen eingesetzt um Tests auszuführen & Ergebnisse zu dokumentieren
- Ziele der Testausführungswerkzeuge:
  - Kostenreduktion
  - Durchführen von weiteren Tests
  - Durchführung der selben Tests in unterschiedlichen Umgebungen
  - Tests leichter wiederholbar machen
  - Tests ausführen, die manuell nicht ausgeführt werden könnten (z.B. umfangreiche Prüfungen der Datenvalidierung)
- Diese Ziele überschneiden sich und lassen sich in den Hauptzielen zusammenfassen: Steigerung des Überdeckungsgrades bei gleichzeitiger Reduzierung der Kosten

7.2.3.1. Anwendbarkeit

- Rentabilität dann am höchsten, wenn diese zur Automatisierung von Regressionstests eingesetzt werden
- Gründe: geringer Wartungsaufwand und Testwiederholungen
- Auch Smoke-Tests kann effektive Anwendungsmöglichkeit sein, aufgrund der häufigen Ausführung der Tests und weil das Ergebnis schnell benötigt wird
- Obwohl bei dieser Anwendung die Wartungskosten höher sein können → automatisierte Methode für Bewertung neuer SW-Versionen in kontinuierlicher Integrationsumgebung
- Testausführungswerkzeuge werden überwiegen in den System- und Integrationsteststufen eingesetzt
- Manche Werkzeuge – insbesondere API-Testwerkzeuge – können auch im Komponententest eingesetzt werden

7.2.3.2. Grundlagen der Testausführungswerkzeuge

- Testausführungswerkzeuge führen eine Reihe von Anweisungen in einer Programmiersprache aus (→Skriptsprache)
- Anweisungen für das Werkzeug sind sehr detailliert und spezifizieren Eingaben, Reihenfolge der Eingaben, bestimmte Eingabewerte, erwartete Ausgaben
  → Dadurch sehr sensibel auf Änderungen des Testobjektes, v.a. bei Benutzung von GUIs
- Die meisten Testwerkzeuge haben einen Testkomparator, der die tatsächlichen mit den gespeicherten Werten vergleicht

7.2.3.3. Testautomatisierung implementieren

- Trend bei automatisiertem Testen - und auch Programmieren – geht weg von detaillierten Anweisungen hin zu höheren Sprachen
- Es werden auch hier Bibliotheken, Makros und Subroutinen benutzt
- Bei Testentwurfsverfahren, wie beim schlüsselwortgetriebenen oder aktionswortgetriebenen Testen, werden Folgen von Anweisungen mit einem bestimmten Namen (Schlüsselwort / Aktionswort) gekennzeichnet → ermöglicht dem TA, die Testfälle in natürlicher Sprache zu erstellen, ohne die zugrundeliegende Programmiersprache und Detailfunktionen zu beachten
- Durch die sehr modularen Testskripte wird die Wartung bei Änderungen an der Funktionalität oder an der Schnittstelle der zu testenden SW erleichtert
- Bestimmung von Schlüsselwörtern kann auf Modellen basieren, z.B. auf Geschäftsprozessmodellen, die häufig in den Anforderungsdokumenten enthalten sind
- So lassen sich die wichtigen Geschäftsprozesse bestimmen, die getestet werden müssen
- Danach können die Schritte dieser Prozesse bestimmt werden, inkl. der Entscheidungspunkte im Verlauf des Prozesses
- Die Entscheidungspunkte können in Aktionswörter umgewandelt werden, welche die Testautomatisierung aus den Schlüsselwort- bzw. Aktionsworttabellen erkennen und verwenden kann
- Geschäftsprozess-Modellierung ist eine Methode zur Abbildung von Geschäftsprozessen und kann eingesetzt werden um die wichtigen Prozesse und Entscheidungspunkte zu identifizieren
- Modellierung kann manuell oder mit Werkzeugunterstützung durchgeführt werden
- Bei werkzeugunterstützter Modellierung dienen die Geschäftsregeln und Prozessbeschreibungen als Eingabequellen

7.2.3.4. Den Erfolg der Testautomatisierung verbessern

- Testfälle müssen auf Automatisierungsfähigkeit untersucht werden: Lohnt sich eine Automatisierung?
- Optimal ist, wenn jeder gegebene Satz von Testfällen manuelle, halbautomatische, automatische Tests enthält
- Nutzen, die durch Automatisierung entstehen:
  - Zeitabschätzung einfacher
  - Regressionstests in späteren Projektphase schneller und sicherer
  - Status und technische Kompetenz der Tester wird höher
  - Besonders hilfreich bei iterativen und inkrementellen Entwicklungslebenszyklen
  - Abdeckung bestimmter Testarten besser → umfangreiche Datenvalidierungsaufgaben
  - Kosteneffektivität höher bei grossen Datenmengen (Eingaben, Konvertierung, Vergleich)
- Mögliche Risiken:
  - Unvollständiges, ineffektives, inkorrektes manuelles Testen automatisieren, ohne Optimierung
  - Testmittelwartung schwierig, wenn SW geändert wird
  - Sinkende Fehlerfindungsrate
  - Fähigkeiten der Tester für effektive Automatisierung womöglich unzureichend

  - Automatisierung von irrelevanten Tests, nur weil diese existieren und stabil sind
  - Mit zunehmender Stabilisierung der SW können Tests überflüssig werden
    → Wiederholungen haben keine Wirksamkeit, Pesticide Paradoson
- Nicht ratsam, manuelle Tests 1:1 zu automatisieren, sollten für Automatisierung neu definiert werden
  → Testfälle formatieren, wiederverwendbare Muster berücksichtigen, fest programmierte Eingabewerte durch Variablen erweitern
- Viele Testausführungswerkzeuge können Testfälle verbinden, Gruppieren, Wiederholen, Reihenfolge kann geändert werden und sie bieten bessere Analyse- und Berichtsfunktionen
- Viele Testautomatisierungswerkzeuge setzen Programmierkenntnisse voraus
- Grosse Testsuiten sind oft schwierig zu aktualisieren und zu managen, wenn nicht mit Sorgfalt entworfen
  → Geeignete Schulungen für die Anwendung der Werkzeuge, Programmierung, Entwurfsverfahren wertvoll
- Bei Testplanung sollte Zeit eingeplant werden, um automatisierte Testfälle regelmässig auch manuell auszuführen, damit das Wissen nicht verloren geht, wie der Test funktioniert und um korrekte Arbeitsweise zu verifizieren sowie um Gültigkeit von Eingabedaten und deren Überdeckung zu prüfen

7.2.3.5. Schlüsselwortgetriebene Testautomatisierung

- Schlüsselwörter / Aktionswörter werden meist dazu verwendet, übergeordnete Geschäftsinteraktionen mit einem System zu benennen (z.B. «Auftrag stornieren»)
- Jedes Schlüsselwort bezeichnet eine Reihe detaillierter Interaktionen mit dem zu testenden System
- Testfälle werden als Sequenzen von Schlüsselwörtern (inkl. relevante Testdaten) spezifiziert
- Bei Testautomation werden die einzelnen Schlüsselwörter als ein oder mehrere auszuführende Testskripte implementiert
- Die Werkzeuge lesen die mit Schlüsselwörtern erstellen Testfälle und rufen die entsprechenden Testskripte auf, die die Funktionalität des Schlüsselworts implementieren
- Testskripte sind sehr modular implementiert, damit sie sich leicht auf bestimmte Schlüsselwörter abbilden lassen
- Wichtigste Vorteile der schlüsselwortgetriebenen Testautomatisierung:
  - Experten für bestimmten Anwendungs- oder Geschäftsbereich können die Schlüsselwörter definieren → macht die Spezifikation der Testfälle effizienter
  - Eine Person, die vorwiegend Fachexpertise hat, kann von der automatischen Testfalldurchführung profitieren (nachdem die Schlüsselwörter als Testskripte implementiert sind), denn sie braucht den zugrundeliegenden Automatisierungscode nicht zu verstehen
  - Testfälle, die unter Verwendung der Schlüsselwörter geschrieben wurden, sind leichter wartbar, da sie mit geringerer Wahrscheinlichkeit modifiziert werden müssen, wenn sich Details der zu testenden SW ändern
  - Testfallspezifikationen sind unabhängig von ihrer Implementierung. Die Schlüsselwörter können mit verschiedenen Skriptsprachen und Werkzeugen implementiert werden

- Automatisierte Skripte (→ Automatisierungscode), welche die Schlüssel- / Aktionswörter verwenden → Entwickler &/ TTA, TA ist zuständig für Erstellung und Pflege der Schüsselwort-bzw. Aktionsdaten
- Schlüsselwortgetriebene automatisierte Tests → Erstellung des Codes in Integrationsphase, Durchführung in Systemphase
- Bei iterativen Vorgehensweisen → Entwicklung der automatisierten Tests = fortlaufender Prozess
- Nach Erstellung der Schlüsselwörter & Eingabedaten sind TA für die Ausführung der Tests & Analyse der dadurch aufgedeckten Fehlerwirkungen zuständig
- Wenn Anomalie entdeckt wird, muss TA die Ursache der Fehlerwirkung untersuchen, um herauszufinden ob das Problem durch die Schlüsselwörter, Eingabedaten, autom. Testskripts, getestete Anwendung verursacht wird
- Erster Schritt bei Fehleranalyse: manuelle Durchführung des betroffenen Tests mit den gleichen Daten → betrifft die Fehlerwirkung die Anwendung selbst?
- Nächster Schritt: Prüfung der Reihenfolge der Tests → Problemursprung zu früherem Zeitpunkt, z.B. durch Produktion inkorrekter Daten?
- Falls Fehlerursprung nicht gefunden wird → Übergabe der Info der Fehleranalyse an TTA / Entwickler

7.2.3.6. Gründe für das Scheitern der Testautomatisierung

- Gründe: Unzureichende Flexibilität bei Verwendung des Testwerkzeugs, mangelnde Programmierfähigkeiten im Testteam, unrealistische Erwartungen hinsichtlich der Probleme, die durch Testautomatisierung gelöst werden können
- Jede Testautomatisierung – genau wie jedes andere SW-Entwicklungsprojekt – nimmt in Anspruch: Management, Aufwand, Fähigkeiten und Aufmerksamkeit
- Für die Erstellung einer nachhaltigen Systemarchitektur ist die Befolgung geeigneter Entwurfsmethoden nötig
- Für das Konfigurationsmanagement und für den Einsatz guter Programmiertechniken muss Zeit bereitgestellt werden
- Die automatisierten Testskripte müssen getestet werden, da sie wahrscheinlich Fehler enthalten
- Performanz muss ggf. optimiert werden
- Benutzbarkeit der Werkzeuge muss berücksichtigt werden, nicht nur für Entwickler, sondern auch für die Personen, die die Werkzeuge zum Ausführen der Skripte benutzen werden
- Es kann notwendig sein, eine Schnittstelle zwischen Werkzeug und Benutzer zu entwerfen, die Zugriff auf die Testfälle ermöglicht und für Tester logisch aufgebaut ist
- Diese Schnittstelle muss trotzdem die vom Werkzeug benötigte Zugänglichkeit bieten