SQS - Zusammenfassung

Def - Software Qualität

Nicht einheitlich definiert. Gesamtqualität setzt sich aus vielen einzelnen Aspekten zusammen.
Hohe Qualität ist oft Kontextabhängig und subjektiv
Komplexität auf 2 Ebenen → Welche Produkteigenschaften Qualität definieren und welche Ausprägung diese haben.

Nach IEEE-730 erfolgreiche Validierung und Verifikation
„Randbemerkung: Verifikation stellt die Frage ob das Produkt korrekt entwickelt wurde. Validation fragt ob generell das richtige Produkt gebastelt wurde"
(& Neuerer ISO/IEC 25010)

Verifikation

Im Bezug stellt Verifikation sicher, dass die Software gemäß Spezifikation und Anforderungen entwickelt wurde.
Hier wird die Korrektheit des Entwicklungsprozesses geprüft
Ziel ist Fehler in frühen Phasen zu erkennen / zu vermeiden

Methoden:
- Code - Reviews
- Inspektion
- Walktrough
- Statische Tests

Kurz gesagt:
Validierung: Haben wir das Richtige gemacht?
Verifikation: Haben wir es richtig gemacht?

Validierung

Prüft ob Erwartungen und Bedürfnisse erfüllt wurden. Wurden die Richtigen Anforderungen implementiert? Funktioniert es wie vorgesehen. Sichergestellt wird, dass die Software ihren Zweck erfüllt.

Methoden
- Funktionale Tests
- Integrationstests
- Abnahmetests
- Benutzertests

Qualitätsfaktoren ISO/IEC 25010

Nach ISO ein umfassendes Regelwerk zur Definition und Bewertung Projektumfeld und Domäne steuern stellenweise.
ISO Definiert: Qualitativ und quantitativ messbar
Helfen bei Spezifikation von Anforderungen
Ermöglichen Beurteilung des Erfüllungsgrads
Unterteilung in zwei Untergruppen

Funktionale Qualität

Äußere Sicht auf ein System. Externe Systemmerkmale.
Definiert "was"? Es geht dabei um Vollständigkeit, Korrektheit Angemessenheit. Bezug auf funktionale Anforderungen

Strukturelle Qualität

Innere Sicht auf ein System. Interne Systemmerkmale und Eigenschaften. Definiert "wie gut". Bezug auf nicht-funktionale Anforderungen wie:

Zuverlässigkeit – Fehlertoleranz, Wiederherstellbarkeit, Ausfallsicherheit
Benutzbarkeit – Verständlichkeit, Erlernbarkeit, Attraktivität, Bequemlichkeit
Wartbarkeit – Modularisierung, Wiederverwendbarkeit, Stabilität.
Portabilität – Adaptierbarkeit, Austauschbarkeit
Effizienz – Ressourcenverbrauch, Zeitverhalten
Kompatibilität – Interoperabilität, Co-Existenz
Sicherheit – Datenschutz, Integrität

Definition - Software Qualitätssicherung

Umfasst Maßnahmen und Methoden, welche auf Überprüfung und Überwachung der Software Qualität abzielen. Ebenso die Sicherstellung ob und zu welchem Grad die Qualitätsfaktoren erfüllt sind. Ist ein kontinuierlicher Prozess.

Klassifikation

Teilung in 3 Kategorien:

(konstruktiv) Organisatorische Methoden: Standards, Templates, Setup, Wissensmanag.

analytisch { Dynamischen Methoden: Testen im Run, Dynamische Analyse
Qs Maßnahmen { Statischen Methoden: Reviews, statische Codeanalyse.

Ziel ist Prävention von Problemen, Identifikation von Mängel und die kontinuierliche Überwachung.
Ergänzung durch Sichweisen für einander.

Fehlerkosten

QS-Maßnahmen so früh wie möglich einleiten, begleitend zu den Phasen eines SW-Projektes.
Je später Fehler identifiziert werden, umso teurer wird es.

Qualitätskosten

Es stellt sich gegenüber: Was kostet die Qualität im Gegensatz zu Fehlern. Hierbei sind Prüfkosten planbar, Fehlerkosten nicht. Die Balance ist hier sehr wichtig. 100% Proof ist nahezu unmöglich und übertriebene Qualitätsprüfungen sind teuer. Ist irgendwann nicht mehr wirtschaftlich.

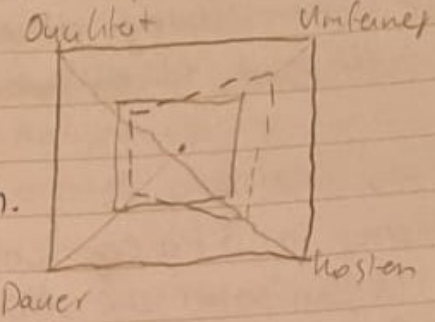
Beeinflusst werden 3 Faktoren: Qualität, Zeit, Kosten

- Good + Fast = Not Cheap
- Good + Cheap = Not Fast
- Fast + Cheap = Not Good

Verzerrung einer Dimension zu Lasten einer anderen.

Teufelsquadrat nach Sneed

Erweiterung des Dreiecks durch aufspalten von Qualität.
Abspaltung des Umfangs.
Vergrößert Handlungsoptionen
"Qualität kann unter Reduktion des Umfangs gehalten werden" → Gilt umgekehrt auch.



Vorgehensmodell

Es gibt hier kein Universalrezept, aber passende Modelle für bestimmte Projekte. (je nach Projekt)
Sequenzielle Modelle (Wasserfall, V-Modell) → Änderungen schwierig
Iterative/Inkrementelle Modelle (Spiralmodell, V-Modell XT) → Wiederholte Durchläufe
Agile Modelle (Scrum / Kaban) Rasche Korrekturen / Änderungen.

Einbettung

Qualität ist ein kontinuierlicher Prozess. (kann kaum am Ende hinzugefügt werden). Erfordert systematisches Vorgehen. Die QS-Aktivitäten sind in jedem Vorgehensmodell vorhanden.

V-Modell (Dinosaurier)

Basis moderner Modelle. Eine Art Links/rechts Abgleich.
Jede konstruktive Phase hat eine konspirative Phase.
Bezieht Verifikation und Validation mit ein
Risiko der Fehlentwicklung durch späte Validierung bleibt

Scrum

Repräsentative für agile SW-Entwicklung
Leichtes Rahmengerüst und iterative Entwicklung in Zyklen.
QS nicht separat. Aktivitäten teil des Sprints (Zyklus)
Teams sind cross-funktional und selbstorganisiert

#2

Anforderungsanalyse

Eine Anforderung ist eine genau beschriebene Notwendigkeit bzw Bedürfnisse, welche gewünscht werden.
Diese sollten den SMART-Kriterien entsprechen

S Specific (präzise/eindeutig)
M Measurable (messbar)
A Achievable (erreichbar/erstrebenswert)
R Realistic (machbar)
T Time-Related (innerhalb gewisser Zeit erreichbar)

Die Klassifikation geht dabei von 2 Arten aus. Funktionale Anforderung, welche angeben was das System können muss und mitunter wie. Nicht-Funktionale mit der Frage, unter welchen Bedingungen.

Gute Anforderungen sind „Vollständig" und „Konsistent" ↳
Das heißt es fehlen keine relevanten Details und die Beschreibung hat keinen Widerspruch

QS von Anforderungen

Hier gibt es die konstruktiven Ansätze wo Methoden bzw Prozesse genutzt werden um während der Erhebung bereits Fehler zu vermeiden (CheckListen, Templates, Workshops). Weiters existieren auch analytische Ansätze, also jene Maßnahme, welche bereits spezifizierte Anforderungen begutachten (Reviews, Def: Abnahmetests)

Statische Qualitätssicherung

Ausführbarkeit nicht nötig. Fokus auf innerer Struktur, Systematische Überprüfung von Artefakten. Variation hinsichtlich Formalität, Zweck, Flexibilität und Rollen möglich. Soll Fehler und Widersprüche sowie Redundanzen und Abweichungen früh identifizieren. Verbessert Produktivität und die Wissensverteilung. Stellt gemeinsame Verantwortung her, sowie eine Konformität. Gut wenn keine Werkzeuge dafür da sind Reviews nach Kost jedem Arbeitsresultat möglich

Erfolgsfaktoren

Qualität und sachliche Kommunikation (sei professionell)
Review des Objektes, vertrauliche Athmosphäre, keine Anschuldigung
Festlegung klarer Ziele, def messbare Bewertungskriterien.
Festhaltung + Nachvollziehbarkeit von Ergebnissen
Passende Review-Typen wählen.
Unterstützung und Akzeptanz

Review-Typen

Resultieren aus Praxis oder Forschung, wobei traditionelle eher aus Forschung kommen und moderne aus der Industrie (praxis).
Bei Traditionellen gibt es klare Rollen und Prozess, vorgegebene Lesetechniken, sie sind schwergewichtig, meeting zentriert und formalisiert. Hingegen moderne leichter, toolgestützt und kontinuierlich sind, sowie Asynchronität und verteilte Teams erlauben
Traditionelle sind im Einklang mit sequenziellen Vorgehensweisen moderne mit agilen Praktiken.
Es gibt 4 Hauptarten

Inspektion: sehr formal, für Fehlerentdeckung und Prüfung auf erfüllte Spezifikation. Def Rollen und ein formaler Bericht. Setzt Fachexperten ein

Tech. Review: formal, für Fehlerentdeckung und Prüfung von tech. Artefakten auf Konformität, sowie Einhaltung von Standards, Regulationen und Guidlines.
Def Rollen + technisch qualifizierte Experten

Walk-Through: wenig formal, für Qualitätsverbesserung und Erwägung von Alternativen. Wissensverteilung (Ausbildung / Einschulung)

Peer-Review: informell, wie Walkthrough, nur mit verschiedener Ausprägungen.

Rollen

- Autor: Urheber des Review-Objektes / Support bei Unklarheit
Moderator: Organisiert und koordiniert, meist QS Experte
Gutachter: Durchführer und präsentiert Resultate. Oft tech Experte
Manager:
Protokollant: Dokumentiert, hält wichtige Erkenntnisse/Entscheidungen fest

Formale Reviews - Prozess

Planung – Def Arbeitspakete und Inhalt. Checklists, Richtlinien, Ziele etc. Teamsetzung und Rollenevaluierung

Kick-off - Reviewstart nach Erteilung der Eingangskriterien, Verteilung ans Team. Ziele und Aufgaben festlegen

Vorbereitung - Review durchführen, intensive Durcharbeitung. Doc von Fragen, Fehlern bzw Kommentaren, Checkliste abarbeiten

Meeting - Resultat präsentieren. Besprechen / Bewerten der Fehler Empfehlungen / Verbesserungsoptionen erarbeiten

Nacharbeit - Mängel beheben, Verbesserungsoptionen umsetzen

Follow-UP - Nachkontrolle. Freigabe von Reviewobjekt bzw neues Review Sammeln von Metriken / Lesson-Learned (Verbesserungsprozess)

Lesetechniken

Hilfestellung für Vorgang bei Review. Sollen Effizienz steigern.

Häufige Ansätze

- Ad Hoc: Kost keine Anleitung, wenig Vorbereitung, Output von Reviewer abhängig
- Checkliste: Ist Reviewbasis → Set an Ja/Nein-Fragen mit Abzug auf Fehlerkategorie
- Perspektiven-basiertes: Reviewobjekt aus mehreren Perspektiven betrachtet um verschiedene Fehlertypen zu finden

Moderne Reviews

Meist bei Source-Code-Review (Weiterentwicklung von Peer-Review)

Asyncron, kontinuierlich, Änderungsbasiert, werkzeuggestützt. (Informell)

Nutzen Funktionen von Versionskontrollsystemen Diff-View / Merge-Request

Geht verglichen mit den traditionellen Reviews schneller, hat häufigere Verbesserungen und benötigt weniger Personen.

Umfang ist kleiner.

Ausprägungen

Pre-Commit: Vor mergen in die gemeinsame Code-Base. Forciert Review-Prozess
Kurze Zeitspanne zw. Entwicklung und Review
Behebungen schnell zu machen → blockieren Abschluss.
kann Entwicklung verzögern

Post-Commit: Review nach Merge. Schnellere Integration, jedoch muss schlechte
Qualität nachträglich verbessert werden und Reviews
könnten ausgelassen werden

Teststufen - Komponententests

Heißen auch Modul-/Unittest und testen einzelne Komponenten isoliert vom Rest des Systems. Test wird von Entwickler geschrieben und benötigt Codezugang

Integrationstests

Für Fehlerzustände in Schnittstellen und Zusammenspiel aufdecken. Werden vom Entwickler geschrieben und haben Fokus auf Integration mehrerer Komponenten. Nötig, da isoliert alles gehen könnte

Systemtests

Testet integriertes System um sicherzustellen, dass es die Anforderungen erfüllt. Testumgebung sollte Abbild der Produktivumgebung sein. Von unabhängigem Testteam ausgeführt.

Akzeptanztests (Abnahmetest)

Testet ob System bereit für produktiven Einsatz ist und die Abnahmekriterien des Kunden erfüllt. Soll keine Fehler finden. Von Kunden / User durchgeführt.

Black-Box-Methoden (Testentwurfsverfahren)

Test auf Basis der Spezifikation, kein Wissen über die innere Struktur. Daten-getrieben (Variation der Eingabe, Abgleich von Erwartung mit dem Resultat). Hierzu gibt es verschiedene Methoden, „Motto: So viele wie nötig, so wenige wie möglich"

Äquivalenzklassen

Einteilung von Inputs in Klassen, wo der gleiche Output erwartet wird. Sollten für gültige, wie ungültige Werte existieren. Damit reicht ein Repräsentant pro Klasse. Häufig mit Grenzwertanalyse gemischt

Grenzwertanalyse

Erhebung von Grenzwerten erleichtert Verhaltenstests. Bei den Grenzwerten sind am Häufigsten Fehler zu finden.

Entscheidungsbasierte Tests

Für Abdeckung komplexer Regeln. Berücksichtigt Abhängigkeiten zwischen Eingabewerten. Vollständiges Testen ist nicht möglich → Entscheidungstabellen helfen bei der Reduktion der Kombinationsmöglichkeiten. Sie stellen die Bedingungen und die resultierende Aktion dar.
Weiters existiert auch der Ursache-Wirkungs-Graph welcher Zusammenhänge / Kausalitäten zwischen Eingaben und Ausgaben darstellt

Zustandsübergangstests

Darstellung des Systemverhaltens durch Zustände. Zeigt normalerweise nur gültige Übergänge, wobei jeder Pfad einem Testfall entspricht

Anwendungsfallbasiertes testen

Ableitung durch Anwendungsfälle, wobei jeder Fall ein Szenario der Systeminteraktion beschreib. Diese Anwendungsfälle enthalten Vor- und Nachbedingung sowie das erwartete Ergebnis.
Abdeckung = $\frac{\text{abgedeckte}}{\text{alle}}$

White Box Methoden (Überdeckungstests)

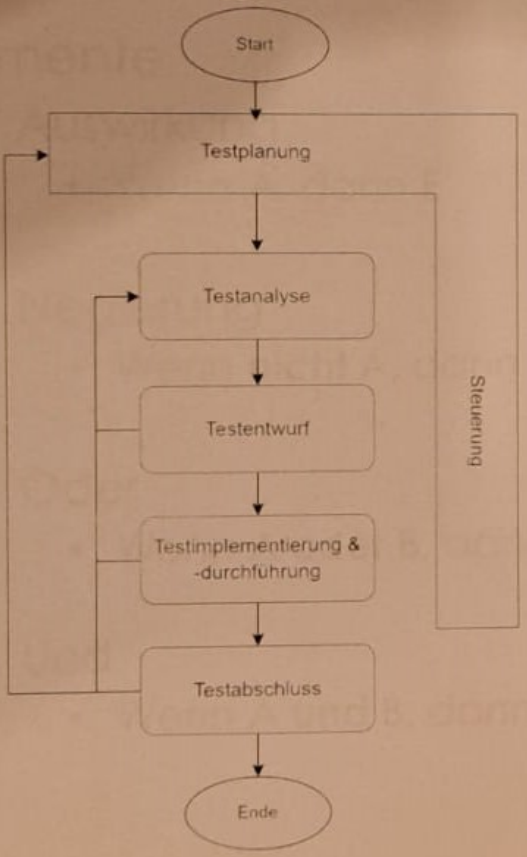
kontrollflussbasiert und strukturorientierte Testmethoden.
Basis ist Analyse des Quellcodes. Orientierung anhand Kontrollflussgraphen. Relevant bei Unit-Test Ebene. Keine Regeln für Testdatenerzeugung definiert. Für Ermittlung der Testabdeckung

Anweisungsüberdeckung: Statement Coverage. Jedes Statement 1x durchlaufen, wobei voll(s)tändige Überdeckung hier erreicht ist, wenn alle Statements durchlaufen werden → Zeigt toten Code. Kriterium ist zu schwach

Zweigüberdeckung: Branch-Coverage (umfasst Statement-Coverage). Jede Kante muss 1x durchlaufen werden. Hilft zu optimieren und zeigt unausführbare Programmteile. Jede Zweigstelle 1x true / 1x false. Keine Rücksicht auf komplexe Logik und Schleifen zu schwach getestet

Pfadüberdeckung: Path-Coverage deckt alle Pfade von Start bis Ende ab. Jede Kombination an Kanten von Start bis Ende 1x durchlaufen. Kaum durchführbar bei komplexen Softwaremodulen. Zu viele Pfade (Schleifen?!)

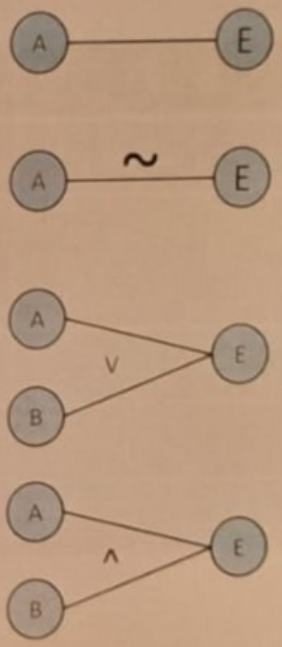
ISTQB fundamentaler Test Prozess



- Testen ist ein Prozess, keine einmalige Aktivität
- Testausführung ist der sichtbarste Teil des Testens
- Um effektiv und effizient zu sein, müssen auch Tests geplant, vorbereitet und ausgewertet werden
- Aktivitäten können überlappend/gleichzeitig stattfinden
- Der Testprozess muss an die Anforderungen des Projekts angepasst werden

Ursache-Wirkung Graph

- Elemente
  - Auswirkung
    - Wenn A, dann E
  - Negierung
    - Wenn **nicht** A, dann E
  - Oder
    - Wenn A **oder** B, dann E
  - Und
    - Wenn A **und** B, dann E



Entscheidungstabelle Beispiel

	1	2	3
Ticket verloren	N	N	Y
Sicherheitscode gültig	Y	N	
Zutritt	Y		
Sicherheitsdienst		Y	
Kein Zutritt			Y

Bedingungsüberdeckung Condition Coverage. - Überprüfung zusammengesetzter und Teilentscheidungen.
Einfacher Test prüft alle atomaren Teilentscheidungen gegen true und false
Mehrfach-Test prüft alle Wahrheitswertkombinationen der atomaren Teilentscheidung.
Sind sehr aufwändig

Für die Coverage reicht ca. 80% aus, jedoch ist sie allein kein ausreichendes Kriterium. Es gilt quantitative Metrik vs. qualitative Methodik
Es ist wichtiger was getestet wird, als wie viel

Erfahrungsbasiertes Testen
Auch Explorativ Testen. Ist mit verschiedenen Testentwurfsverfahren kombinierbar. Die Testfälle basieren auf Intuition und Erfahrung des Testers. Wo sind Probleme wahrscheinlich, wo gab es schon viele? Was wurde schnell gemacht?
Muss dokumentiert werden

Test Doubles #5

Vergleichbar mit Stunt Double. Austausch von komponente gegen eine andere alternative Implementierung zu Testzwecken

Dummy Objekt

Leerer Platzhalter ohne Funktionalität. Dient für Vervollständigung von Komponenten, wenn die gereichten Dummies nicht zum Einsatz kommen. (Leeres Objekt)

Fake Objekt

Ausführbare Implementierung. Man stellt damit das Verhalten fest. Wie wird reagiert? BSP Simulierte Datenquelle. Wenn das echte Teil nicht verfügbar ist oder zu langsam wäre.

Spy

Erweiterung einer Komponente um eine Kontrollfunktion. Reale Implementierung muss in der Funktion gleich bleiben. Kann sämtliche gewünschte Eigenschaften protokollieren.

Stub(s)

Ausführbare Implementierung, welche vordefinierte Werte/Exceptions an den Aufrufer. Responder-Stub (Ernstfall / Saboteur-Stub (Error Code) Steuert welcher Pfad durchlaufen wird und erlaubt das Testen solcher.

Mock

Exceptions
↓
Ausführbare Implementierung mit vordefinierten Fehlern/Werten Interaktionen zwischen Testobjekt und Mock überwacht, übergebene Parameter geprüft, sowie gegen das erwartete Verhalten geprüft Ist selbst Teil des Tests und erkennt unerwartete werte.
↳ Assertions bzw Exceptions

Continuous Integration

Kommt aus Extreme Programming. Änderungen werden laufend integriert, getestet und gebaut. Integrationen durch automatisierten Buildingprozess

Ausgestaltung

Prinzipiell häufige Commits und Builds. Stabile Builds.
Ist natürlich: Projektabhängig bzw. von Technologischen Gegebenheiten abhängig. Ebenfalls auch vom gewünschten Umfang

Vorteilhaft dabei sind schnelles Feedback, sowie wenige manuelle Tätigkeiten. Fehlererkennung ist früh möglich und das Refactoring ist ebenfalls erleichtert → Vermeidet Integrationshölle und das Problem „bei mir funktionierts". Erhöhte Transparenz von Status und Qualitätsniveau
Fokus liegt darauf, auto. Build bis zum ersten lauffähigen SW-Stand zu machen.

Aufbauende Ansätze

Continous Delivery (CDE) → Automatisierung von Packaging-/Releaseprozess
Continuous Development - CCD)
→ Automatisierung des Deployment-Prozesses
→ Autom. Bereitstellung neuer SW-Versionen

Relevanz für DevOps

Trennung zwische Entwicklung (Produktbereitstellung) und Betrieb (Deployment). Problem ist geringe Rücksicht auf Betrieb in der Entwicklung und fehlendes Testen betrieblicher Aspekte. Weiters auch spätes Finden von Blockern und Fehlende Transparenz und Know-How

Refactoring

Systematischer Ansatz um Code-Qualität zu steigern. Soll Lesbarkeit / Verständlichkeit verbessern. Verringert Komplexität und verbessert das Design. Soll keine Funktionalität oder Fehler verändern / beheben bzw nichts Ändern, was die Codestruktur nicht verbessert

Technische Schuld

Schlechter Code = sich in Schuld begeben
↳ Schulden müssen zeitnahe beglichen werden

- Code verwahrlost mit der Zeit → keiner kümmert sich drum
- Manchmal bewusst Schuld begangen um kurzfristig schnellere Resultate liefern zu können.

Anhäufung von Mängeln führen zu Produktivitäts- und Wartbarkeitsverlust, sowie zu hohen Fehlerraten und Fehlerkosten

Vorgehen

Identifikation → Verbesserungsoptionen finden
Testabdeckung → Sicherstellen dass Codeteil gut getestet ist
Durchführen → Schrittweise verbessern → immer wieder testen

Bad-Smells

Indikatoren für Refactoring, welche in Gruppen unterteilt sind
Bloaters: Zu groß gewordener Code (Klasse, Methode, Parameterliste)
Object-Orientation-Abusers: Falsche/unvollständige Nutzung des Paradigma
Couplers: Zu hohe Klassenabhängigkeit
Change-Preventers: Änderungen bei X haben Auswirkungen bei vielen anderen
Dispensables: überflüssiger Code

Refactoring – Patterns

Gruppe	Fokus	Beispiele
Composing Methods	Struktur/Schnitt von Methoden	• Extract Method • Extract Variable • Replace Temp with Query
Moving Features between Objects	Verschieben von Funktionalität zwischen Klassen	• Move Method • Move Field • Extract Class
Organizing Data	Verwaltung/Kapselung von Daten	• Encapsulate Field • Replace Magic Number
Simplifying Conditional Expressions	Vereinfachen von logischen Ausdrücken	• Decompose Conditional
Simplifying Method Calls	Vereinfachen von Methodenaufrufen	• Rename Method • Add Parameter
Dealing with Generalization	Erzeugen von Vererbungshierarchien Verschieben von Funktionalität zwischen Vererbungsstrukturen	• Pull Up Field • Pull Up Method • Extract Superclass

#6

Agile Softwareentwicklung

Im traditionellen Projekt-Management ist ein Top-Down ansatz häufig. Projektmanager sind verantwortlich und der Kunde liefert Anforderungen, Projektmanager weißt Tätigkeiten zu. Tests und QS spät im Projekt, Späte Lieferung an kunden. Manager soll Unsicherheiten beseitigen und Vorhersagbarkeit erzwingen

Agile Softwareentwicklung legt fest, dass Individuen und Interaktionen mehr als Prozesse und Werkzeuge sind, sowie funktionierendes Software mehr als umfassende Dokumentation, Zusammenarbeit (Kunde) mehr als Vertragsverhandlung und Reaktionen mehr als Befolgen eines Plans

12 Prinzipe

- Frühe und kontinuierliche Auslieferung
- Anforderungsänderungen auch spät willkommen
- Funktionierendes regelmäßig ausgeliefert
- Fachexperten und Entwickler arbeiten täglich zusammen
- Team braucht Umgebung und Unterstützung + Vertrauen
- Beste Weitergabe von Informationen in Person
- Fortschritt anhand funktionierender Software gemessen
- Nachhaltige Entwicklung durch konstante Arbeitsgeschwindigkeit
- Technische Exzellenz und gutes Design durch durchgehende Aufmerksamkeit
- Selbstorganisierte Teams ideal
- Regelmäßige Reflexion im Team für Effektivitätsverbesserung

Extreme Programming

Kundennähe und iterative Entwicklung. Requirements als Stories und eine einzelne Code-Base. Pairprogramming und inkrementelles Design. Test First Programming und CI

SCRUM

Iterativer Prozess in Sprints mit fixer Zeitdauer unterteilt. Definition of Done sagt welches Maß an Qualität benötigt wird.

Sprints

Alle 1-4 Wochen. Beginnt damit, dass die umzusetzenden Items geschätzt und festgelegt werden. Daily Scrum Meeting für interne Abstimmung. Endet mit Sprint Review bzw Retrospective. Sprints können nur abgebrochen aber nicht verlängert werden

Kaban

Pull-System und Leichtgewichtig, um Einführungswiderstand zu minimieren, Kaban + Scrum = Scrumban
Virtualisiert Workflow und Limitiert Work-in-Progress.
Regeln des Prozesses und gemeinsame Verbesserung

Agiles Testen

Baut auf ~~dem~~ Testmitstand auf. QS von Beginn an, Fokus auf Kommunikation und Zusammenarbeit. Steruhre Entwicklung im ganzen Lebenszyklus involviert. Wissen und Erfahrung für das Projekt. Risikoidentifikation im Projekt. Unterstützt Analysten bei Definition von User-Stories und Akzeptanztests

Test Driven Development (TDD)

Entwicklung der auto. Test vor Produktiv-Code-Entwicklung. Verbesserter Design im Code. Sicherheitsnetz bei Codeänderung für Entwickler. Frameworks für viele Sprachen verfügbar
Test erstellen → Test Failure → Product Code → Test green → Refactor

Behaviour-Driven Development (BDD)

RED
Gemeinsame Entwicklung von (Beispielen) Szenarien wie das System funktionieren soll. Auch als Living Documentation bezeichnet.
Szenario make → Exec Fail → Code make → Exec Success → Refactor
Szenario: Formulierung
Glue Code: Übersetzung in Aufrufe an das Testsystem
System under Test: System / Funktionalität die zu testen ist

Advanced TDD (ATDD)

Es werden Akzeptanztests zu User-Stories definiert und Test natürlichsprachlich formuliert. Akzeptanztest nicht zwingend auto. Ähnlich BDD aber höheres Abstraktionslevel

Testautomatisierung

UI-Tests oder e2e (end to end) Tests.

- Waren früher nicht tech. unterstützt und häufig „false positive".
Heute weniger, durch Tools, aber immer noch möglich.
Erfordert Abstimmung von Regeln zwischen Entwicklern und Testern. Positivfall testen. Setup-Abkürzungen benutzen → direkt REST statt UI. Auf Sauberkeit und Struktur im Code achten

Agile Testquadranten nach Crispin & Gregory

Einteilung der Testtypen in Quadranten mit gleicher Priorität

Q1: Technische Test (Teamsupport)

Unittests entwickeln. Regelmäßige Integration und Testausführung
Zeigen interne Produktqualität.
Werkzeuge: Git (e.g. Äq), IDEs, Apache Maven/etc, CI-Server

Q2: Geschäftsorientierte Tests (Teamsupport)

Auto.-Akzeptanztests (von nicht-Entwicklern verstehbar)
Überscheidung mit Q1 möglich, Q2 höheres Abstraktionslevel
Zeigen ext. Qualität und ob man „done" ist
Werkzeuge: Checklists, Tabellen etc → manuell, BDD → auto.

Q3: Geschäftsorientierte Tests (Produktkritik)

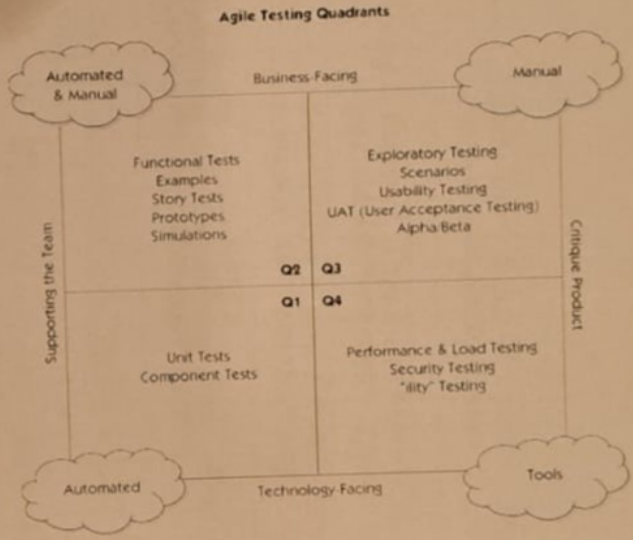
Primär manuell wie explorative Tests, Szenario basiertes testen bzw Session basiertes. Auch Usability Tests, Testet vor allem Schnittstellen

Q4: Technische Test (Produktkritik)

Testen nicht-funktionaler Anforderungen.
Braucht oft Spezialisten. Performance und Last Test dabei

Agile Testquadranten nach Crispin & Gregory

- Einteilung der verschiedenen Test-Typen in unterschiedliche Quadranten
- Keine Priorisierung oder Reihung der einzelnen Quadranten => es werden alle Quadranten benötigt



Q2	Q3
Q1	Q4

Quelle: Lisa Crispin, Janet Gregory: Agile Testing: A Practical Guide for Testers and Agile Teams, Addison-Wesley Professional, 2008

Testversuch

Welche Aufgaben hat Qualitätsmanagement? Wo liegen die Unterschiede zwischen QM und QS

QM konzentriert sich auf strategische Ausrichtung und das gesamte System. Es werden Qualitätskriterien und -Ziele festgelegt.

Planung, Überwachung/Steuerung, Verbesserung, Risikomanagement und Personalbelegung sind Kernthemen.

QS hat operativen Fokus und dient dazu, dass konkrete Produkte auf Erfüllung der Qualitätsanforderungen zu prüfen

3 Schwerpunkte SQM

- Qualitätsplanung: Def. von Qualitätszielen (Performance, etc.) Auswahl geeigneter Standards, Tools und Methoden
- Qualitätssicherung: Reviews, Test, Inspektionen um Erfüllung der Anforderungen sicher zustellen
  Identifizieren und Beheben von Fehlern während der Entwicklung
- Qualitätskontrolle und Verbesserung:
  - Überwachung der Einhaltung von Qualitätsstandards
  - Einführen von Verbesserungsprozessen

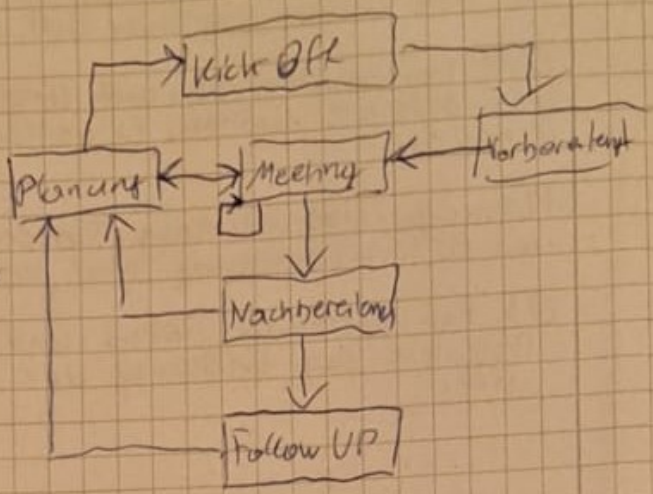
Was ist ein Review? Unterschied von Review und Test? Vor- und Nachteile?

Ein Review ist eine Art Rückblick auf eine getane Arbeit. Hier wird begutachtet was und wie etwas gemacht wurde. (Nicht zwingend Code).
Der Unterschied zu Tests ist, dass ein Review auch andere Aspekte beleuchtet. So kann ein Test zwar feststellen ob die Funktionalität gegeben ist, jedoch kann nicht festgestellt werden ob der Code bspw gut lesbar, wartbar oder erweiterbar ist.
Der Vorteil am Review ist, dass Diskrepanzen und andere Fehler wie Verständnisprobleme ausgeräumt werden können, bevor sie großen Schaden bzw Aufwand verursachen.
Nachteilig ist, dass sie selbst Aufwand erzeugen → kosten Zeit und Geld

Wie ist ein Review aufgebaut

Planung: Def der Arbeitspakete und Reviewinhalte Rollen festlegen

Kickoff: Initierung nach Erfüllung der Eingangskriterien Zielsetzung und Aufgabenverteilung



Vorbereitung: Durchführung, intensive Durcharbeitung Dokumentieren von Fehlern, Unklarheiten etc.

Meeting: Präsentation der Ergebnisse, Besprechung und Bewertung der identifizierten Fehler Erarbeiten von Verbesserungsoptionen

Nachbereitung: Behebung der Mängel, Umsetzen der Verbesserungsvorschläge

Follow-UP: Nachkontrolle, Freigabe des Reviewobjekts Sammlung von Metriken

4 Arten von Testdoubles + Beschreibung und Vor/Nachteile

Dummy-Objekt: Leeres und funktionsloses Objekt. Nur für Vervollständigung von Komponenten im Test

Fake-Objekt: Vereinfachtes Objekt für Vervollständigung bspw simulierte Datenquelle, wenn real-Implementation zu langsam

Spy: Komponentenerweiterung um Kontrollfunktion. Reale Implementierung muss in Funktionsweise gleich bleiben.

Stub: Realimplementierung mit vordefinierten Werten. Liefert diese bei Aufruf zurück und kann geprüft werden.

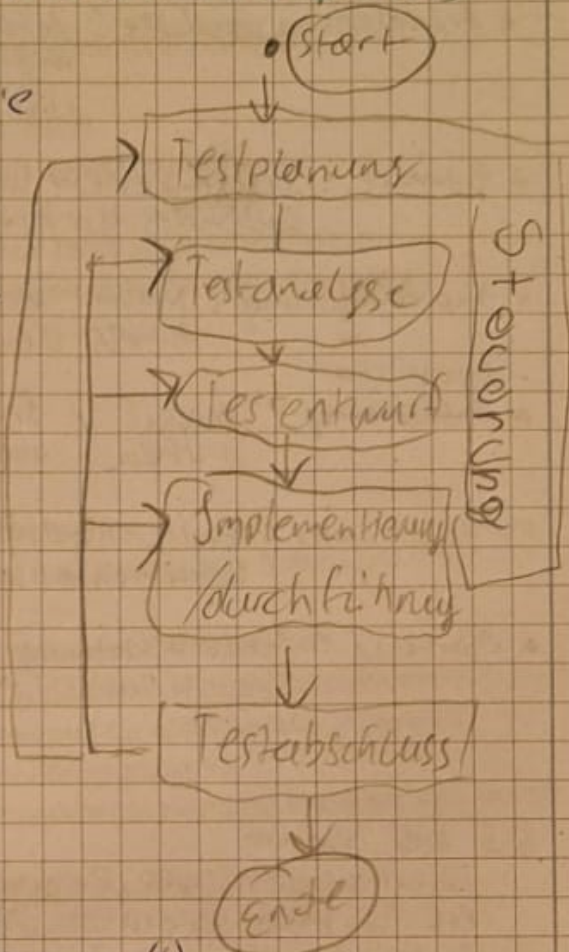
Genereller Vorteil bei TestDoubles, dass leichtere Implementierungen möglich sind und bestimmtes Verhalten leicht erzwungen werden kann

Refactoring in Softwareengineering, wann und wie?

Was ist das

Refactoring meint eine Art Restrukturierung des Codes um die Qualität zu erhöhen ohne dabei die Funktionalität zu verändern. Test sollten nach Refactoring durchlaufen! Eingesetzt wird es nachdem die Funktionalität gewährleistet ist (laut Spezifikation). Der Code wird lesbarer, effizienter wartbarer, etc.

Nennen und beschreiben Sie die Phasen im Testprozess



Testplanung: Festlegung von Teststrategie und Zielsetzung sowie Ressourcen und Zeitplan (Umfang und Testarten)

Testanalyse: Bewertung der Testergebnisse und Freigabeentscheidung Analyse von Fehlerursache und Qualitätsmängeln

Testentwurf: Erstellen von Testszenarien Testfällen und Testdaten, sowie identifizieren von Testbedingungen und Erwartungshaltung.

Testvorbereitung: Testinfrastruktur herstellen und Testdaten zur Verfügung stellen

Testdurchführung: Implementieren und durchführen des Tests (automatisch oder manuell). Dokumentieren der Ergebnisse

Testabschluss: Archivieren von Testartefakten und Erfahrungswerte (Lesson Learned) dokumentieren um Testprozess zu verbessern. Freigeben der Testumgebung und Ressourcen

2 Prozessmodelle in der Softwareentwicklung nennen und anhand eines modells erklären, wie dann Qualitätssicherung gewährleistet wird

Wasserfallmodell
Agile Entwicklung → Scrum

QS im Wasserfallmodell ist eine spezifische Aufgabe in jeder Phase des Modells. Findet häufig spät statt

- Anforderungsanalyse: Anforderungsspezifikationen erstellen und prüfen, mittels Review Korrektheit, Vollständigkeit und Eindeutigkeit sicherstellen.
- Entwurf: Design-Review und Modell-Inspektion. Sicherstellen, dass Systemdesign gemäß Anforderungen und Prinzipien
- Implementierung: Nutzen der Code-Standards und Best-Practices, Mittels Reviews und Code-Analyse prüfen
- Integration
- ~~Auslieferung~~: Modul-, Integration-, Systemtests und Funktionalität prüfen.
- Auslieferung: Endgültige Test vor Freigabe. Prüfung der Dokumentation und Schulung von Anwendern
- Wartung: Fehlerbehebung + regelmäßige Updates. Regressionstest → Sicherstellen dass nicht kaputt gemacht wurde

QS bei Scrum
Leichtgewichtiges Rahmengerüst und iterative Entwicklung. QS ist kein separater Prozess sondern eng mit Prinzipien und Praktiken der SW-Entwicklung gekoppelt. QS-Aufgabe ist hier Teil des Sprints.

Test 1 Fragen

1. Welchen Kontext bei Qualitätsmetriken kennen Sie? Nennen Sie mindestens drei.
2. Nennen Sie die Phasen des SPI-Zyklus nach Deming!
3. Welche vier Zielsetzungen, die sich gegenseitig beeinflussen, werden bei Softwareentwicklung sichtbar?
4. Nennen Sie die Phasen im Test-driven Development!
5. Nennen Sie die Elemente einer User Story Card!
6. Teststufen den Beschreibungen zuordnen (Komponententest, Integrationtest, Systemtest, Akzeptanztest, Lasttest, Regressionstest)
7. Refactoring: Was ist das, welche Schritte macht man?
8. Ablauf eines Reviews von Anforderungen erklären und skizzieren.
9. Nennen und beschreiben Sie 4 ISTQB Testprinzipien.
10. 4 Merkmale/Unterschiede zu BlackBox WhiteBox Tests gegenüberstellen.
11. Äquivalenzklassenzerlegung und Grenzwertanalyse anhand eines Beispiels beschreiben.
12. Verifikation und Validierung erklären.
13. 6 Leistungen der QS-Stelle aufzählen
14. 4 Arten von Messungen nennen und beschreiben
15. Was ist eine Inspektion?
16. Welche Teststufen gibt es? Nennen und beschreiben.
17. Was ist Testautomatisierung? Welche Ziele hat sie? In welchen Teststufen kann sie angewendet werden?
18. Beschreiben Sie Testen in SCRUM sowie 5 Aufgaben von TesterInnen.
19. Was ist der Unterschied zwischen Anweisungsüberdeckung und Zweigüberdeckung? Geben Sie je ein Beispiel (hier sollte man je ein Beispiel für eine Anweisungs- und Zweigüberdeckung geben, z.B. durch einen Kontrollflussgraphen)
20. 3 verschiedene typische Stakeholder eines Projekts nennen und erklären was durch die
"typische" Haltung schiefgehen kann (Kunde weiß nicht was er/sie will..)
21. QS in verschiedenen Vorgehensmodellen erklären
22. Vorteile von Testautomatisierung

1. Welchen Kontext bei Qualitätsmetriken kennen Sie? Nennen Sie

mindestens drei.

- Produkteinsatz
- Produktänderung
- Produktwechsel

2. Nennen Sie die Phasen des SPI-Zyklus nach Deming!

Plan: Planen der nächsten Iteration des SPI-Zyklus basierende auf aktuellen Erfahrungswerten (inkl. Zielsetzung)
Do: Durchführen der geplanten Aktivität (Task, Projekt, usw).
Check: Überprüfung der Produkt- und Prozessergebnisse (z.B. durch Messung)
Act: Analyse der Ergebnisse als Feedback für den nächsten SPI Zyklus. Verbesserung der Erfolgsfaktoren, Vermeidung von Wiederholungsfehlern, Beseitigen von Schwachstellen usw.

3. Welche vier Zielsetzungen, die sich gegenseitig beeinflussen, werden **bei Softwareentwicklung sichtbar?**

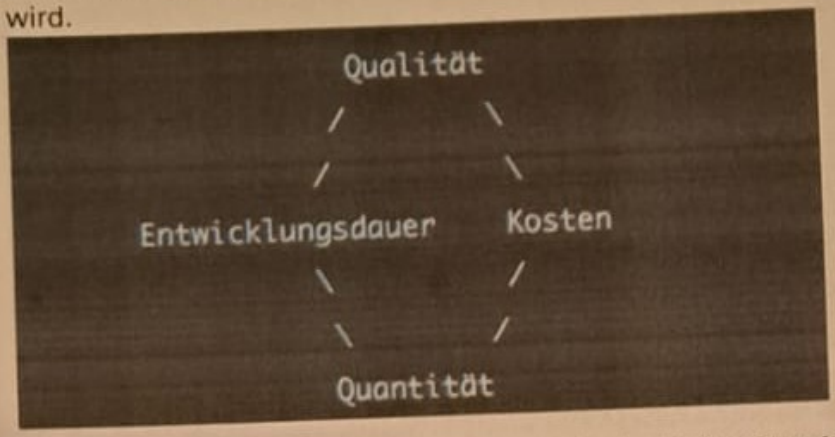
Qualität
Beispiele: Anzahl der Fehler in einem Produkt, Termin- und Kostentreue.
Quantität
Beispiele: die Anzahl der implementierten Funktionen, Function Points, Lines Of Code (LOC).
Entwicklungsdauer (z.B. in Monaten) bzw. Aufwand (z.B. in Personenmonaten)
Beispiele: kleine“ vs. „große“ Projekte
Kosten

4. Nennen Sie die Phasen im Test-driven Development!

Think
Auswahl der zu implementierenden Anforderungen.
Spezifikation der Testfälle.
Red: Implementierung und Ausführung der Testfälle
Alle Tests müssen fehlschlagen.
Green: Implementierung der Komponenten und Klassen und Ausführung der Testfälle
Testfall erfolgreich → weiter bei Schritt 4.
Testfall schlägt fehlt → weiter bei Schritt 3.
Refactor: Änderung und Optimierung der Implementierung ohne Änderung der Funktionalität; Ausführung der Testfälle.
Testfälle dürfen nicht fehlschlagen.
Auswahl der nächsten Anforderung (Schritt 1)

5. Nennen Sie die Elemente einer User Story Card!

Card: Die Karte (Card) ist das physische Medium, das die User- Story beschreibt. Hier werden die Anforderung, ihre Dringlichkeit, die erwartete Dauer für Entwicklung und Test sowie die Abnahmekriterien für diese Story identifiziert. Die Beschreibung muss genau sein, da sie im Product Backlog verwendet wird.
Conversation: Die Diskussion (Conversation) erklärt, wie die Software genutzt werden wird.



Confirmation: Die Abnahmekriterien, die in der Diskussion festgelegt werden, werden verwendet, um den Abschluss einer User-Story bestätigen (confirm) zu können.

6. Teststufen den Beschreibungen zuordnen (Komponententest,

Integrationtest, Systemtest, Akzeptanztest, Lasttest, Regressionstest)
Komponententest: Testen einzelner Komponenten der Software
Integrationstest: Testen der Integration von mehreren Komponenten oder Modulen der Software
Systemtest: Testen des Gesamtsystems in seiner Umgebung
Akzeptanztest: Testen der Anforderungen durch den Kunden oder Nutzer
Lasttest: Testen der Leistungsfähigkeit der Software unter hoher Last
Regressionstest: Testen der Software nach Änderungen, um sicherzustellen, dass keine Regressionen aufgetreten sind.

7. Refactoring: Was ist das, welche Schritte macht man?

Refactoring ist ein Prozess der Verbesserung der Softwarequalität, bei dem der Code umstrukturiert wird, um die Lesbarkeit, Wartbarkeit oder Effizienz zu verbessern. Die Schritte umfassen:
Identifizierung von Verbesserungsmöglichkeiten
Planung der Umstrukturierung
Umsetzung der Umstrukturierung
Überprüfung der Wirksamkeit der Umstrukturierung

8. Ablauf eines Reviews von Anforderungen erklären und skizzieren.

Planung: Objekt, Prüfziele, Auslösekriterien (Einstiegskriterien), Teilnehmer, Ort, Zeit.
Vorbesprechung: Vorstellung des Prüfobjekts bei komplexen und neuen Produkten.
Intensive Einzeldurcharbeitung
Durchführung: Gemeinsames Lesen, Aufzeichnung von Mängeln; während des Reviews sollen Mängel entdeckt, nicht korrigiert werden.
In der Nachbearbeitung werden dokumentierte Mängel korrigiert und in der Bewertung überprüft.
Berichterstattung.
Wiederholungen von Reviews sind möglich.
Checklisten unterstützen Reviews.
Typische Dauer: 2h

9. Nennen und beschreiben Sie 4 ISTQB Testprinzipien.

#1 Testen zeigt Fehler auf
Beweist aber nicht, dass keine Fehler mehr vorhanden sind
Black Box Tests WhiteBox Tests
Basiert auf Spezifikation Basiert auf der Struktur von Code bzw.
Modulen.
Kein Wissen über innere Struktur
vorhanden
Wissen über innere Struktur erforderlich
Daten-getriebener Test Logik-getriebene Tests
Anforderungsüberdeckung Überdeckung von Knoten, Kanten, Pfaden
Partitionierung der Eingabedaten Partitionierung von Daten für
Bedingungsüberdeckung
#2 Vollständiges Testen ist nicht möglich
Sämtliche Kombinationen von Eingaben, Vorbedingungen und Zuständen zu testen ist nicht umsetzbar/praktikabel
Stattdessen wird der Testfokus mittels Risiken und Prioritäten festgelegt
#3 Frühzeitig Testen
Testaktivitäten sollten so früh wie möglich begonnen
werden
Frühzeitig gefundene Fehler sind einfach und günstig zu beheben
#4 Fehlerhäufungen beachten
Fehler sind nicht gleichmäßig verteilt. Werden in einem Modul einige Fehler gefunden, sind dort meist weitere zu erwarten
#5 Veränderung statt Wiederholung
Das wiederholte Ausführen von Testfällen wird keine
#6 Testen ist kontextabhängig
#7 Trugschluss: Fehlerlose Systeme sind brauchbar
Fehler zu finden und zu beheben garantiert nicht, dass das Produkt den Anforderungen und Bedürfnissen des Benutzers entspricht

10. 4 Merkmale/Unterschiede zu BlackBox WhiteBox Tests gegenüberstellen.

Black Box Tests	WhiteBox Tests
Basiert auf Spezifikation	Basiert auf der Struktur von Code bzw. Modulen.
Kein Wissen über innere Struktur vorhanden	Wissen über innere Struktur erforderlich
Daten-getriebener Test	Logik-getriebene Tests
Anforderungsüberdeckung	Überdeckung von Knoten, Kanten, Pfaden
Partitionierung der Eingabedaten	Partitionierung von Daten für Bedingungsüberdeckung

11. Äquivalenzklassenzerlegung und Grenzwertanalyse anhand eines

Beispiels beschreiben.
Äquivalenzklassenzerlegung: Zerlegung einer Menge von Daten (Input oder Output) in Untermengen (Klassen), die äquivalente Ergebnisse oder Auswirkungen produzieren. Jede
Klasse(nkombination) sollte mindestens einmal getestet werden.
Grenzwertanalyse: Erweiterung der Äquivalenzklassenzerlegung für bessere Überdeckung.
Grenzwerte werden als Klassenrepräsentanten gewählt

12. Verifikation und Validierung erklären.

Verifikation:
„Bauen wir das Produkt richtig?"
Umsetzung im Vergleich zur Spezifikation in vorangegangenen Phasen.
Test der Umsetzung gegen die Spezifikation
Validierung:
„Bauen wir das richtige Produkt?"
Entspricht die Lösung und damit die Spezifikation dem, was der Kunde erwartet?
Test der Umsetzung gegen die Nutzeranforderungen.

13 6 Leistungen der QS-Stelle aufzählen

Qualitätsplanung als Teil der Projektplanung.
Herstellen lokaler Standards auf Projekt- und Organisationsebene.
Review zentraler Projektdokumente.
Organisieren von Reviews: Ausbildung, Planung, Durchführung,
Verbesserungsvorschläge.
Unterstützung bei Personalauswahl und Software-Zukauf.
Vorbereitung und Auswertung von Produkttests.

14. 4 Arten von Messungen nennen und beschreiben

Direkte vs. indirekte Messungen:
Direkte Messung: Wertermittlung direkt beim zu untersuchenden Objekt (z.B. Dauer, Aufwand einer Aufgabe)
Indirekte Messung: Ermittlung von Messwerten aus direkten Messungen (z.B. Effizienz einer Fehlererkennungsmethode = Anzahl der gefundenen Fehler pro Zeiteinheit)
Objektive vs. Subjektive Messungen
Objektive Messung: LoC, Auslieferungsdatum, Aufwand usw.
Subjektive Messung: Messergebnisse basierend auf der individuellen Sichtweise des Betrachters, z.B. Fragebögen zur Erfassung der Kundenzufriedenheit.
Quantitative vs. Qualitative Daten:
Quantitativ: Daten als konkrete Zahlenwerte (z.B. für statistische Auswertungen)
Qualitativ: Visualisierte Informationen (Text, Bilder), z.B. durch Interviews, Interpretationen.

15. Was ist eine Inspektion?

Bei der Inspektion sind die Ziele schwere Defekte im Prüfobjekt zu identifizieren, den Entwicklungsprozess zu verbessern und Metriken zu ermitteln.
Die Teilnehmer sind die Moderatorin, Autor, Gutachterin, Protokollführer, (Vorleser).
Die Charakteristika sind eine ausgebildeter Moderatorin, das Prüfobjekt wird vom Vorleser
Absatz für Absatz vorgetragen. Die Moderatorin gibt die Freigabe für dies.

16. Welche Teststufen gibt es? Nennen und beschreiben.

Komponententest: Testen von einzelnen Modulen oder Komponenten
Integrationstest: Testen der Schnittstellen zwischen verschiedenen Komponenten oder Modulen
Systemtest: Testen des gesamten Systems gegen die Anforderungen
Abnahmetest: Testen des Systems gegen die Benutzeranforderungen und Abnahme durch den Kunden
Regressionstest: Wiederholung von Tests, um sicherzustellen, dass Änderungen keine negativen Auswirkungen auf bestehende Funktionen haben
Last- und Performance-Test: Testen der Leistung und Skalierbarkeit des Systems unter Last

17. Was ist Testautomatisierung? Welche Ziele hat sie? In welchen

Teststufen kann sie angewendet werden?
Testautomatisierung ist der Einsatz von Softwaretools, um Testfälle automatisch auszuführen und Ergebnisse zu überprüfen. Die Ziele sind die Reduzierung von Testaufwand und -kosten, die Erhöhung der Testabdeckung und -effektivität sowie die Verbesserung der Testqualität. Testautomatisierung kann in allen Teststufen eingesetzt werden, insbesondere im Komponenten-, Integration- und Regressionstest.

18. Beschreiben Sie Testen in SCRUM sowie 5 Aufgaben von TesterInnen.

Testen in SCRUM beinhaltet kontinuierliches Testen während des gesamten Entwicklungsprozesses, um sicherzustellen, dass die Anforderungen erfüllt werden und dass das System fehlerfrei ist.
Der Scrum Prozess sieht keine Tester vor.
Testen ist Teil der Entwicklung
Unit Tests (!)
Test Driven Development
Testen geschieht durch den Product Owner / Kunden
Jeden Sprint
Akzeptanzkriterien
Aufgaben eines Testers
Statische Tests der Anforderungen / Dokumentation
Dynamisches Testen (Ausführung von Akzeptanztests)
Erstellung von Akzeptanztests
Sicherstellung der Verwaltung von Testfällen im SCM
Unterstützung bei Unit Tests
Unterstützung bei test-getriebenem Design
Test Ansatz
Test Planung
Regressionstests (manuell & automatisiert)
Input während des Meetings (Planung, Sprint Review, etc.)
Risikoanalyse und –abschätzung
Schätzung der User Stories
Demonstration und Unterstützung während des Sprint Reviews
Tracability von Tests zu User Stories
Erstellung und Wartung von automatisierten Tests
Unterstützung bei Akzeptanztests

19. Was ist der Unterschied zwischen Anweisungsüberdeckung und

Zweigüberdeckung? Geben Sie je ein Beispiel (hier sollte man je ein Beispiel für eine Anweisungs- und Zweigüberdeckung geben, z.B. durch einen Kontrollflussgraphen)
Anweisungsüberdeckung
Statement Coverage
Ziel: jede Anweisung muss mind. einmal durchlaufen werden.
Vollständige Anweisungsüberdeckung liegt vor, wenn sämtliche Anweisungen mindestens einmal durchlaufen werden.
Zeigt ob toter Code existiert
Anweisungen, die niemals durchlaufen werden können
Zu schwaches Kriterium für sinnvolle Testdurchführung
Zweigüberdeckung
Branch Coverage
Umfasst Anweisungsüberdeckung vollständig
Ziel: Jede Kante muss mind. einmal durchlaufen werden
Zeigt nicht ausführbare Programmzweige auf
Hilft oft durchlaufene Programmteile gezielt zu optimieren
Im Gegensatz zum Anweisungsüberdeckungstest muss jede Entscheidung mindestens einmal true und false annehmen
Problematik:
Unzureichender Test von Schleifen
Komplexe Logik in Statements wird nicht berücksichtigt

20. 3 verschiedene typische Stakeholder eines Projekts nennen und

erklären was durch die "typische" Haltung schiefgehen kann (Kunde weiß

nicht was er/sie will..)

Der Kunde: Er/Sie hat eine bestimmte Vorstellung davon, was das Produkt leisten soll und welche Anforderungen es erfüllen muss. Wenn der Kunde jedoch nicht genau weiß, was er/sie will oder unklare Anforderungen stellt, kann dies dazu führen, dass das Team in die falsche Richtung arbeitet oder das Produkt nicht den Erwartungen des Kunden entspricht.
Der Projektleiter: Er/Sie hat die Aufgabe, das Projekt zu planen und zu koordinieren und sicherzustellen, dass es innerhalb des Budgets und des Zeitrahmens abgeschlossen wird. Wenn der Projektleiter jedoch nicht genug Ressourcen zur Verfügung hat oder unklare Anforderungen vom Kunden erhält, kann dies dazu führen, dass das Projekt verzögert wird oder nicht den Anforderungen entspricht.
Das Entwicklerteam: Es ist verantwortlich für die Umsetzung des Produkts und muss sicherstellen, dass es den Anforderungen des Kunden entspricht. Wenn das Team jedoch nicht genug Informationen über die Anforderungen hat oder unklare Anforderungen erhält, kann dies dazu führen, dass das Team Zeit damit verbringt, den Anforderungen auf den Grund zu gehen, anstatt produktiv zu arbeiten.

21. QS in verschiedenen Vorgehensmodellen erklären

Die Rolle der QS (Qualitätssicherung) hängt stark vom Vorgehensmodell ab. In einem Wasserfallmodell wird die QS oft am Ende des Entwicklungsprozesses durchgeführt, um sicherzustellen, dass das fertige Produkt den Anforderungen entspricht. In agilen Vorgehensmodellen wie Scrum wird QS jedoch während des gesamten Entwicklungsprozesses durchgeführt, um sicherzustellen, dass das Produkt kontinuierlich
verbessert und den Anforderungen des Kunden entspricht.

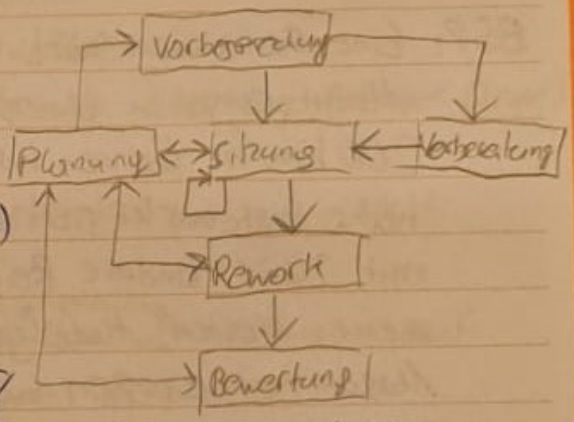
22. Vorteile von Testautomatisierung

Effizienz: Tests können automatisch ausgeführt werden, was Zeit und Ressourcen spart.
Wiederholbarkeit: Tests können automatisch wiederholt werden, um sicherzustellen, dass das Produkt konsistent und zuverlässig ist.
Skalierbarkeit: Automatisierte Tests können leicht auf verschiedene Plattformen und Umgebungen skaliert werden.
Zuverlässigkeit: Automatisierte Tests sind genau und reproduzierbar, was dazu beiträgt, Fehler zu identifizieren und zu beheben.
Früherkennung von Fehlern: Durch automatisierte Tests können Fehler frühzeitig im Entwicklungsprozess erkannt und behoben werden, bevor sie zu teuren Problemen werden.

① Produkteinsatz, Produktwechsel, Produktänderung
② Do, Check, Act
③ Kosten, Qualität, Entwicklungs, Quantität
④ Think, Red, Green, Refactor (Test make, Fail, Code make, Succes, Refactor)
⑤ Card, Conversation, Confirmation
⑥ Komponententest: Einzeln und isoliert getestete Komponente
Integrationstest: Zusammenspiel zwischen Komponenten
Systemtest: Lauffähigkeit des „Ganzen" und störungen unter bestimmten Voraussetzungen zu finden
Akzeptanztests: Dienen eigentlich als Testfälle bei der Abnahme.

⑦ Verbesserungsprozess um Qualität zu steigern. Verbesserung von Lesbarkeit, Effizienz und Wartbarkeit.
Schritte: Möglich Verbesserungen finden
Planen der neuen Struktur
Umsetzen
Funktionalitätserhalt prüfen, sowie Wirksamkeit des Neuen

⑧ Planen: wann, wo, was, warum · reviewer!
Vorbesprechung: Objekt vorstellen
Vorbereitung: Individuell durcharbeiten
Durchführung: Sitzung mit allen Beteiligten, Lesen, ev. Aufzeichnen von Mängeln (noch nicht fixen)
Rework: Verbesserung der Fehler
Bewertung: korrekturen geprüft
Wiederholen von Reviews ist möglich!
Checklisten sind gutes Hilfsmittel um sauber zu arbeiten



⑨ #1 Testen zeigt Fehler auf → Sagt nicht, dass keine mehr da sind
#2 Vollständiges Testen ist unmöglich → 100% kaum umsetzbar und nicht praktikabel
#3 Frühzeitiges Testen → Testen so schnell wie möglich einbinden um Zeit, Geld und Mühe zu sparen.
#4 Fehlerhäufungen betrachten → Wo ein paar sind können auch noch mehr sein. Dort genauer hinschauen

(10)

Black	White
- Spezifikationsbasiert	- Basis ist Codestruktur bzw Module
- Innere Struktur unbekannt	- Innere Struktur bekannt
- Datengetrieben	- Logikgetrieben
- Partitionierung von Eingabedaten	- Überdeckung von Knoten, Kanten pfaden
- Anforderungsüberdeckung	- Datenpartitionierung für Bedingungs-überdeckung

(11)

Äquivalenzklassen: Zerlegung von Datenmengen in Klassen. Eine Eingabe einer Klasse hat immer das gleiche Resultat. Eingaben unterschiedlicher Klassen haben unterschiedliche Resultate

Grenzwertanalyse: Testen mit allen Werten einer Klasse ist zu aufwändig. Es reicht die Werte an den Übergängen / Grenzen zu testen.

BSP: Eine Reifenwerkstatt verkauft Reifen + Montage. Dabei macht es einen Unterschied ob ein Auto schneller als 200 km/h fahren kann und ob es Stahl oder Alu-Felgen hat. Dies wirkt sich auf den Preis aus, da für Autos mit 200+ andere Reifen benötigt werden, welche 40€ mehr kosten. Leichtmetallfelgen müssen mit einer anderen Maschine bezogen werden, um sie nicht zu beschädigen. Diese Montage dauert länger und kostet daher 10€ mehr. Die Auftragsgebühr beträgt standardmäßig 150€ (ohne Material)

Klasse Felge:
- Alu
- Stahl

Stahl		Alu	
>200	≤200	>200	≤200
150€	180€	160€	200€

Klasse Gesch
- >200
- ≤200

(12) Verifikation: Wurde das Produkt gemäß den Spezifikationen richtig gemacht?

Validierung: Ist das Produkt / Kann das Produkt das was gefordert wurde. Sind die Spezifikationen richtig?

Unterschied besteht, dass Verifikation angibt ob es den Spezifikationen entspricht, während Validierung feststellt ob die Spezifikationen korrekt sind

(13)

Qualitätsplanung bei Projektplanung
Lokale Standards auf Projekt- und Organisationsebene
Review zentraler Projektdokumente
Reviews organisieren
Unterstützung bei Personalauswahl und Software-Zukauf
Produkttest vorbereiten und auswerten

(14) Quantitative → Messung nach Zahlenwerten (wie viel, wie schnell)
Qualitativ → Visualisierte Informationen
Subjektive → Messbasis auf individueller Sicht
Objektiv → LoC, Auslieferungszeitpunkt (Wertung statischer Dinge)

(16) Komponententest, Integrationstest, Systemtest, Abnahmetest
↳ Frage (6)
Regressionstest: Testwiederholung um Änderungen auf negative Auswirkungen zu prüfen
Lost Test: Test unter Belastung

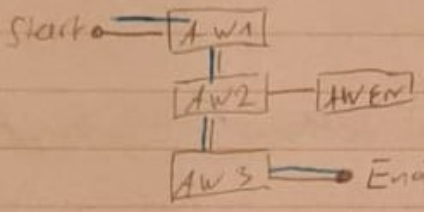
(15) Formell und für Fehleraufdeckung → Prüfung gegen die Spezifikationen. Hat definierte Rollen (Moderatorin, Autor, Gutachter, Protokollführer).

(17) Softwareansatz um Tests automatisch durchzuführen und Ergebnisse zu prüfen. Reduziert Aufwand und Kosten, erhöht Effizienz und Abdeckungen. In allen Teststufen einsetzbar

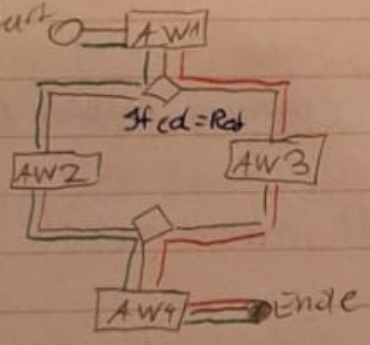
(17) Kontinuierlicher Prozess während der gesamten Entwicklung um Fehler zu finden und auf Abweichungen bei den Spezifikationen zum Produkt zu finden. SCRUM sieht „per se" keine Tester vor.

Aufgaben: Statisches Testen
Dynamisches Testen
Risikoanalyse und -abschätzung
Erstellung und Wartung automatisierter Test
Unterstützung bei Akzeptanztest
Schätzung der User-Stories

(18) Statement-Coverage: Jede Anweisung min. 1* durchlaufen ist das Ziel. → Vollständige Coverage wenn erfüllt. Zeigt ob toter Code existiert. Farbiger Pfad durchläuft alle Anweisungen 1* (AWEn ist als Zweites Ende zu sehen)!



Branch Coverage: Jede Verzweigung ist 1* zu durchlaufen, also jede Kante im Graphen. Zeigt unausführbare Programmteile. Die beiden Farben zeigen die zwei verschiedenen Wege von Start zu Ende. Die Bedingung gibt an welcher Pfad durlaufen wird



(20) Kunde: Hat bestimmte Vorstellunge, weiß aber nicht ganz genau was er will → Man arbeitet in die falsche Richtung

Projektleiter: Plant und koordiniert. Wenn zu wenige Ressourcen vorhanden bzw unklare Anforderungen vorliegen ⇒ Projektverzug

Entwicklerteam: Verantwortlich für Projektumsetzung. Bei zu geringen Informationen kann Team viel Zeit für falsche Dinge verschwenden

(21)

Wasserfallmodell: Rückkehr ist schwer. QS wird am Ende des Entwicklungsprozesses durchgeführt um gegen Anforderungen zu prüfen

Agile Modelle: QS wird während des gesamten Prozesses immer durchgeführt. kontinuierliche Verbesserung

(22) Effizienz → Spart Zeit + Ressourcen
Wiederholbarkeit → Automatische Wiederholung
Skalierbarkeit → Anpassung ist leicht möglich
Zuverlässigkeit → Genau und Reproduzierbar
Früherkennung → Erkennen von Fehlern möglichst früh