SEPM Prüfungsfragen Ausarbeitung

27. Juni 2020

Block 1

Einführung in Software Engineering

Unterschied zwischen *Embedded Systems* und *kommerzieller Software* anhand von fünf Kriterien vergleichen.

Kriterium	*Embedded Systems*	*Kommerzielle Software*
Steuerung	Ereignisgesteuert, oft auch vollständig automatisiert	Benutzergesteuert
Kosten	Teuer, wegen Neuentwicklung oder Anpassung	Kaufen billiger als selber entwickeln
Zuverlässigkeit	Sehr wichtig	Oft nicht entscheidend
Wartung	Schwierig, z.T. Hardwaretechnisch unmöglich	Meist professioneller Support
Sicherheit	Müssen sicher sein (safety)	Unterschiedliche Wichtigkeit: Online-Banking, Datenbank Systeme vs. Photobearbeitung
Usability	Benutzerinterface rudimentär oder nicht vorhanden	Oft entscheidend, vor allem bei großer Konkurrenz
Beispiele	Handysteuerung, Lift-Steuerung, ABS-System, Ampel	Datenbanksystem, Web-Applikationen, Texteditor

Block 2

Einführung in Projektmanagement

Was ist ein Projekt? Durch welche Merkmale ist es definiert?

Ein Projekt ist ein *einmaliges Vorhaben* mit einem definierten *Anfang*, einem definiertem *Ende* und *mehreren beteiligten Personen.*

Welche drei Kennzeichen hat ein Projekt?

- Ihre Abgrenzbarkeit bezüglich: Aufgabe, Ergebnis, Ressourceneinsatz und Zeitrahmen
- Komplexität der Aufgabe
- Einmaligkeit der Aufgabe

Welche Voraussetzungen gibt es für ein Projekt und warum?

Was ist Projektmanagement?

Projektmanagement ist eine *Gesamtheit* von *Methoden* und Verhaltensweisen zur effizienteren *Steuerung* der Abwicklung von besonderen *Aufgabenstellungen* (Projekten).

Im engeren Sinn ist das Projektmanagement die *Projektleitung*, d.h. die für die Führung/Steuerung eines Projekts verantwortliche Person/Stelle.

Welches Vorgehen verwendet man in der Steuerung?

- Die Aufgabenstellung definieren
- Das Projekt planen
- Das Projekt(team) organizieren
- Aufgaben verteilen
- Den Projektfortschritt kontrollieren
- Bei Bedarf steuernd eingreifen

- Sich mit Risiken beschäftigen
- Entscheidungen vorbereiten und Entscheidungen fällen
- Verwaltungskram

Drei typische Gründe warum ein Projekt den Zeit-/Kostenrahmen überschreitet (2019-07-04)

- Mangelhafte Anforderungen

  Die Anforderungen decken sich nicht mit den Erfordernissen oder sind unklar/ mehrdeutig formuliert.

- Mangelhafte Umsetzung der Anforderungen

  Die Anforderungen wurden nicht verstanden und deshalb falsch umgesetzt, es gibt technische Mängel (Implementierung) oder Termin-Not und Ressourcenmangel.

- Mängel im Projektmanagement

  Mangelhafte Kommunikation, unrealistische Termin- und Kostenrahmen oder Ressourcenmangel (Verfügbarkeit, Qualifikation, Erfahrung).

Projektauftrag

Der Projektauftrag ist eine schriftliche Vereinbarung zwischen Auftraggeber und Auftragnehmer, ein Projekt zu bestimmten Bedingungen durchzuführen.

Komponenten vom Projektauftrag

- Projektname
- Projektbeschreibung
- Rollen und Verantwortlichkeiten
- Nicht-Ziele, Abgrenzungen
- Komponentendiagramm
- Wiederferwendung von Komponenten, Technologiewahl
- Lieferumfang und Abnahme
- Nichtfunktionale Anforderungen
- Arbeitsstruktur
- Zeit und Kostenplan

- Informationswesen
- Besonderheiten
- Umfeld- und Risikoanalye

Skizzieren und beschreiben das Teufelsquadrat

Die vier angegebene Ziele des Quadrates konkurrieren um die verfügbare Produktivität, welche durch die Fläche des äußeren Vierecks dargestellt wird. Aufgrund der Begrenzungen der Ressourcen ergibt sich automatisch eine Begrenzung der Produktivität, symbolisiert durch die Fläche des inneren Vierecks. Man kann das Viereck in die eine oder andere Richtung strecken,muss dann allerdings einen geringerem Zielerfüllungsgrad auf der andere Seite hinnehmen.

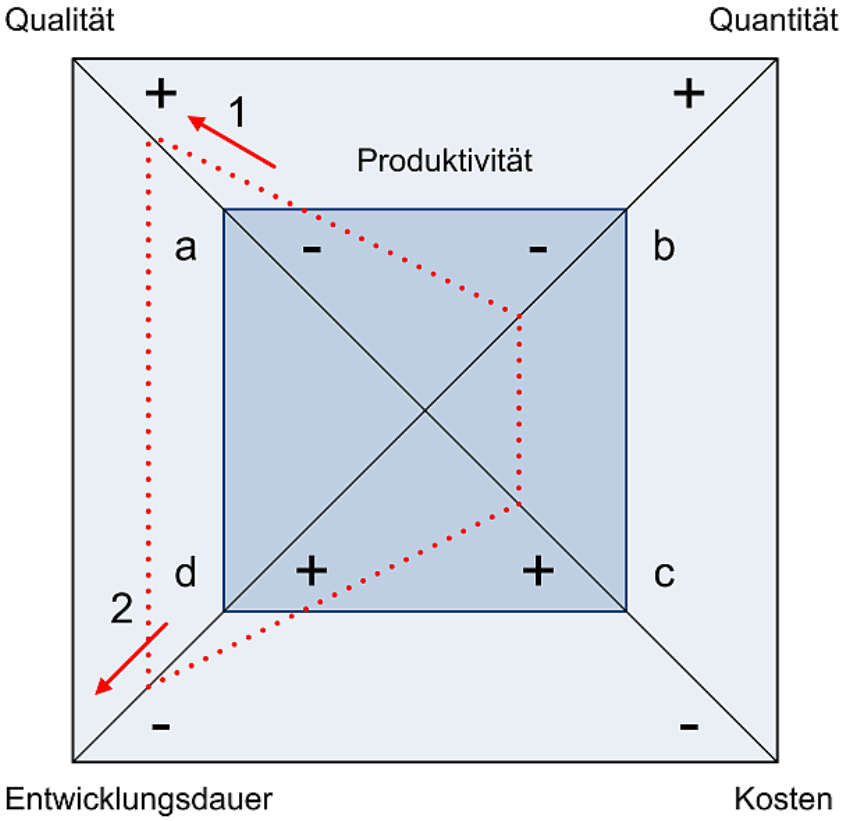


Abbildung 2.1: Teufelsquadrat

Block 3

Technik und Werkzeuge

Was sind die Unterschiede zwischen zentralen und verteilten SCM? Nennen Sie jeweils Vor- und Nachteile (2019-07-04)

Zentralisierte Systeme
Bei zentralen SCM gibt es einen Server, der die Versionen speichert. Alle SCM-Operationen müssen über das Netzwerk an den Server geschickt werden. Daraus ergeben sich auch die Nachteile von zentralen SCM: Der Server ist ein „Single Point of Failure“ und ein Flaschenhals, dadurch können einzelne SCM Operationen sehr lange dauern (hohe Auslastung, Netzwerkverkehr). Dafür benötigen die Clients nicht alle Dateien und Versionen und kommen mit weniger Speicherplatz aus.
Verteilte Systeme
Bei verteilten SCM wird das Repository geklont und lokal auf jedem Client gespeichert wodurch alle SCM Operationen lokal ausgeführt werden können (Vorteil). Der Nachteil hierbei ist das ein große Menge an Daten beim klonen übertragen werden und auf dem Client gespeichert werden muss. Meist gibt es auch in verteilten SCM einen Server mit einem „Main-Repository“ mit welchem sich die Clients regelmäßig synchronisieren (pull, push, fetch, pull-request, etc.)

Große Unterschiede gibt es auch beim Umgang mit Konflikten:

Arbeitsablauf SVN (Zentral)

1. Update
2. Lokale Änderungen
3. Update
    - Änderungen remote und lokal
    - Merge
4. Commit

Arbeitsablauf Git (verteilt)

1. Kopie des Haupt-Repository (klonen)
2. Lokale Änderungen
3. Lokale Commits
4. Merge
    - Push
    - Pull Request

Continuous Integration (CI) beschreiben und skizzieren (2019-07-04)

Continuous Integration Werkzeuge unterstützen Server-basierte Integration und Ausführung von Tests. Diese Automatisierung beinhaltet:

1. Ereignis oder Zeitgesteuerter Abruf
2. Verwendung von Build Werkzeugen
3. Ausführung von Tests
4. Erstellen von Berichten
5. Verschicken von Mitteilungen

Der CI Arbeitsablauf ist in Abbildung 3.1 skizziert und startet wenn ein Entwickler eine neue Version im SCM erstellen (Commit, Pull-Request oder ähnliches, Skizze 1). Das CI Werkzeug reagiert auf dieses Ereignis und lädt die entsprechende Version (Skizze 2). Anschließend wird das Build Tool gestartet und das Kompilat mit einem Test-System automatisiert getestet (Skizze 3). Im letzten Schritt werden die Entwickler verständigt und ein Bericht und das ausführbare Software-Artefakt verteilt (Skizze 4).

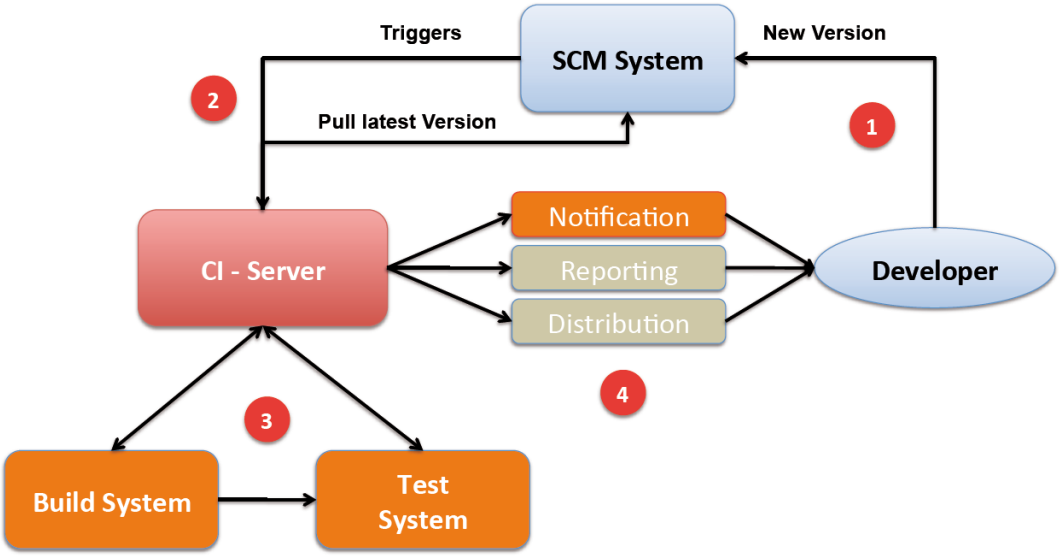


Abbildung 3.1: Ablauf eines Continuous Integration Prozesses

Beschreiben Sie Inversion of Control (IOC).

Abhängigkeiten werden von einem Container verwaltet, die Komponenten wissen nichts darüber (IOC ist transparent). Die Abhängigkeiten werden injiziert (siehe auch Dependency Injection DI). Vorteile von IOC sind:

- Hohe Wiederverwendbarkeit
- Einfaches Austauschen einer Implementierung
- Verwalten von verschiedenen Konfigurationen (dev vs. prod)
- Automatisiertes „verdrahten“, weniger Boilerplate-Code benötigt.
- Verteilung von Aufgaben

Beispiele für Implementierungen/Libraries:

- Java: Spring-Framework, Google Guice, Pico-Container, Apache Aries Blueprint
- .NET: Unity Framework, Spring .NET, Ninject, Autofac

Block 4

Software Engineering Phasen

Skizzieren Sie den Software Life-Cycle.

Der Software Life-Cycle, gezeigt in Abbildung 4.1, beschreibt ein Basiskonzept für Software Engineering Prozesse und Vorgehensmodelle.

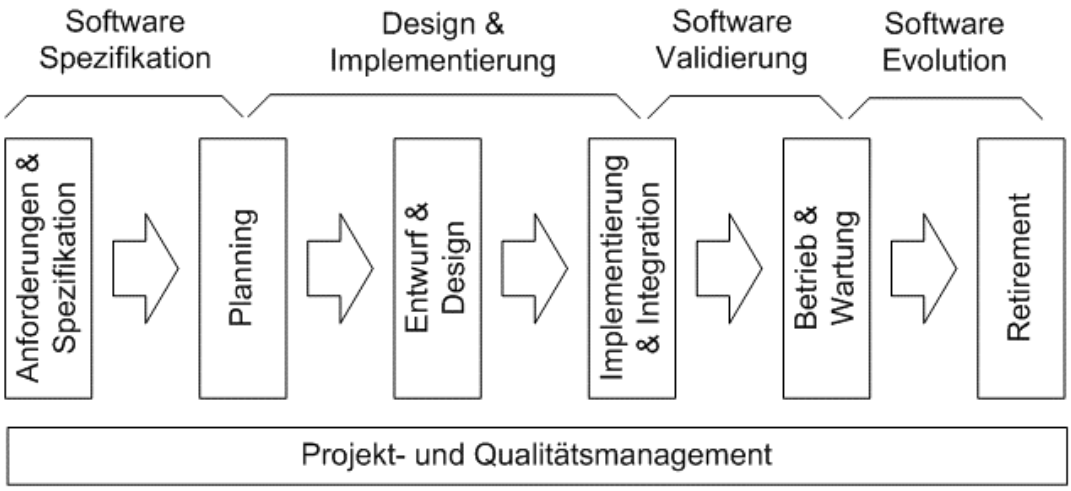


Abbildung 4.1: Der typische Life-Cycle eins Software-Projekts als Basis für Prozesse und Vorgehensmodelle.

Nennen Sie drei Stakeholder, die Einfluss auf die Anforderungen haben. Begründen Sie Ihre Antworten.

- **Kunde:** bezahlen für ein Software. Anforderung z.B. schnelle und kostengünstige Lieferung.
- **Anwender:** müssen mit dem System arbeiten. Anforderungen z.B. Erfüllung von funktionalen und nicht-funktionalen Anforderungen (Usability, Einfachheit, Stability, usw.)
- **Entwickler:** erstellen das Softwareprodukt. Anforderungen z.B. neuester Stand der Technik, neue Funktionalitäten und ihre Umsetzung bewerten.

Klassifizierung von Anforderungen

- **Funktionale Anforderungen** Was bzw. welche Funktionalität soll umgesetzt werden? Wie soll sich das System in definierte Situationen verhalten? Datenformate
- **Nicht-Funktionale Anforderungen** Leistung und Performance, Usability, Sicherheit, Warbarkeit
- **Designbedingungen** worauf soll beim technischen Entwurf geachtet werden, z.B. Schnittstellen, Plattformen und Entwicklungsumgebung
- **Prozessbedingungen**Rahnembedingungen im Softwareprojekt, z.B: Ressourcen/ Dokumentation

4+1 View Model of Architecture

Eine Sicht (view) beschreibt einen Teilaspekt der Architektur und des Designs und die specifischen Eigenschaften eines Systems.

Logical View

- Funktionale Anforderungen
- Fokus auf den End-User
- Beispiele: Design Packages, Subsysteme, Klassen
- UML2: Klassendiagramme, Statemachines, Package Diagrams

Implementation View

- statische Software Komponenten
- Fokus auf Implementierung
- Beispiele: Configuration managemnet, Source Code
- UML2: Component Diagram der vorhandenen Softwareteile

Process View

- Nicht-Funktionale Anforderungen
- Fokus auf Systemintegration
- Beispiele: Laufzeitbedingungen, Concurrency, Lastverteilung, Fehlertoleranz
- UML2:Sequence, Activity Diagrams, Communication Diagrams

Deployment View

- Aufführbare Applikation (zur Laufzeit) unter Berücksichtigung der Plattform
- Fokus auf System Engineers
- Beispiele: Deployment, Installation, Performance
- UML2:Deployment Diagrams

Use-Case View:

- Gemeinsamen Nenner, in dem die Anwendungsfälle und die Aktivitäten (als Szenarien) abbildet und in einen Zusammenhang setzt
- Fokus auf Systemanalyse sowie Enwurf und Design - Schnittstellenfunktion zwischen den anderen Sichten aus der Architektursicht und beinhaltet Schlüsselszenarien (key scenarios) der Applikation aus der Sicht der Geschäftsprozesse
- Fokus auf Implementierung und Transition - Verifikation und Validierung der Anforderungen (Test) und der 4 Architektur-Views
- UML2: Use Case Diagram + Beschreibung, Aktivitätsdiagramme.

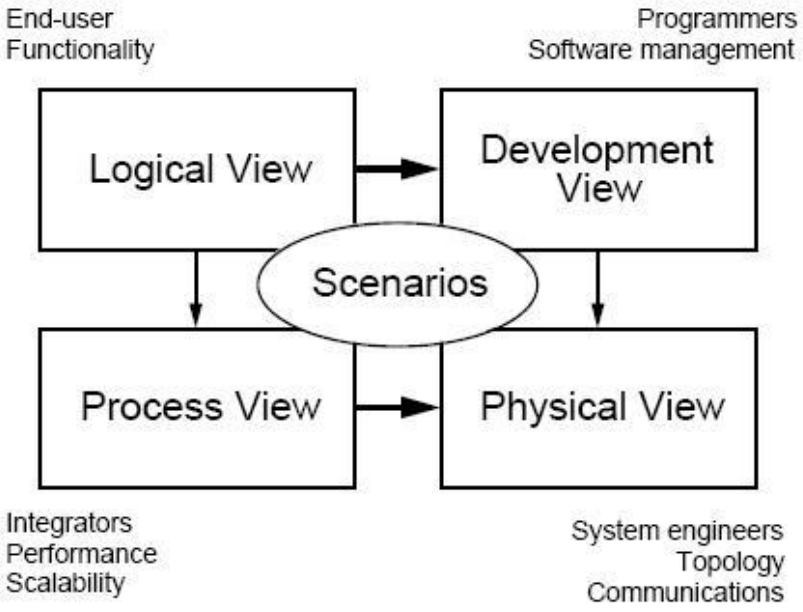


Abbildung 4.2: 4+1 View Model

System Control (Stair vs Fork)

Stair

- Verteilte Kontrolle
- Schrittweise Ausführung von Funktionen, dadurch wechselt die Kontrolle
- Verbesserung der Wiederverwendbarkeit der Methoden z.B. durch Vererbung
- Die spätere Wartung wird erschwert

Fork

- Ein zentrales Objekt kontrolliert den gesamten Use Case
- Wiederverwendung von Datenobjekten wird erleichtert
- Wartbarkeit wird verbessert, da nur ein Objekt geändert werden muss.

Beschreiben Sie Traceability. Welche Arten von Traceability gibt es?

Traceability ist die Nach- oder Rückverfolgbarkeit einer Information durch den gesamten Entwicklungszyklus (z.B. bei sicherheitskritischen Anwendungen gefordert). Sie umfasst auch die Änderungsverfolgung welche den Lebenszyklus einer Anforderung vom Ursprung über die verschiedenen Verfeinerungs- und Spezifikationsschritte bis zur Berücksichtigung in Entwicklungsartefakten nachvollziehbar macht.

Vorteile:

- Nachvollziehbarkeit der Information bei Änderungen

- (Automatische) Benachrichtigung bei Änderungen.

Mithilfe von Traceability können folgende typische Fragestellungen beantwortet werden:

- Woher kommt eine Anforderung und wo wurde sie umgesetzt?
- Welche Artefakte sind von einer Änderung der Anforderung betroffen?
- Welche Anforderungen sind von einer Änderung der Umsetzung betroffen?

Es gibt drei Arten von Traceability:

- **Vertikale Traceability:** Beziehungen innerhalb einer Phase und eines Artefakt-Typs, z.B. System – Subsystsem – Komponente.
- **Zeitliche Traceability:** Zeitliche Nachvollziehbarkeit unterschiedlicher Releases, z.B. durch Konfigurationsmanagement.
- **Horizontale Traceability:** Beziehungen zwischen unterschiedlichen Entwicklungsartefakten, z.B. Anforderungen – Implementierung – Testfälle.

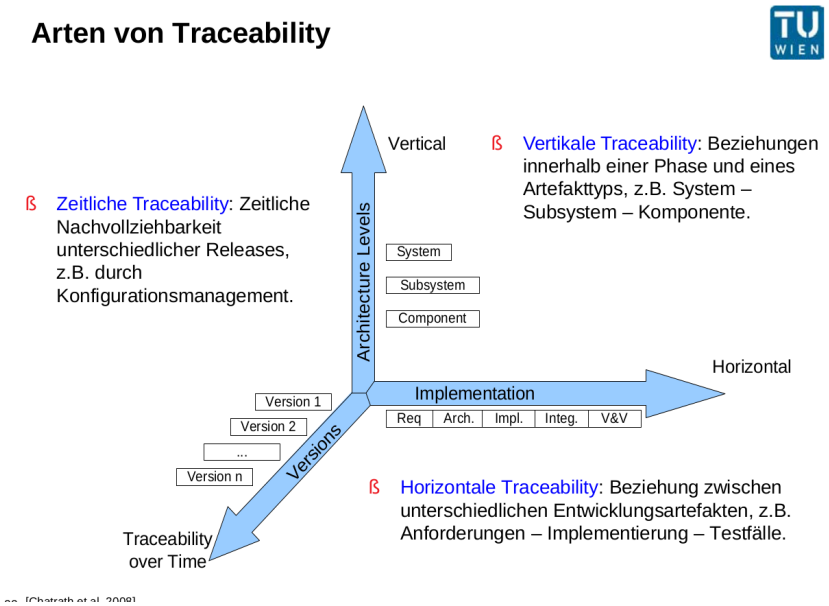


Abbildung 4.3: traceability

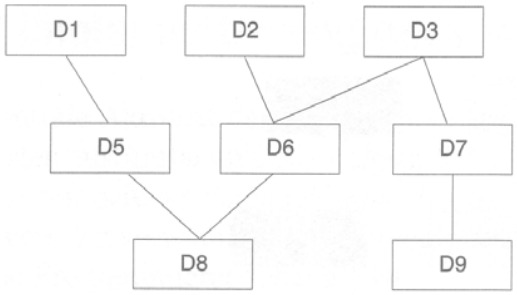
Wie kann Traceability durchgeführt/umgesetzt werden?

Was ist System Integration und welche Modelle gibt es? Erstellen Sie jeweils eine Skizze und nennen Vor- und Nachteile

(Komplexe) Software-Projekte sind aus Gründen der Lesbarkeit, Verständlichkeit und Wartbarkeit meist in (viele) *Module/Komponenten* aufgeteilt.Systemintegration bezeichnet die Integration unterschiedlicher Komponenten und Komponenten zu größeren (Sub-)Systemen.

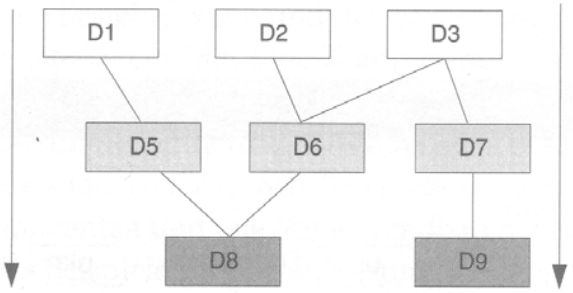
Big-Bang: Alle Module werden *gleichzeitig* integriert (=„Big-Bang“)

- Vorteile: Keine Test-Stubs/Driver notwendig da alle Module bereits verfügbar sind.
- Nachteile: Fehler sind sehr schwer zu lokalisieren, hohes Risiko bei der Integration.
- Anwendung für kleine und überschaubare Projekte.



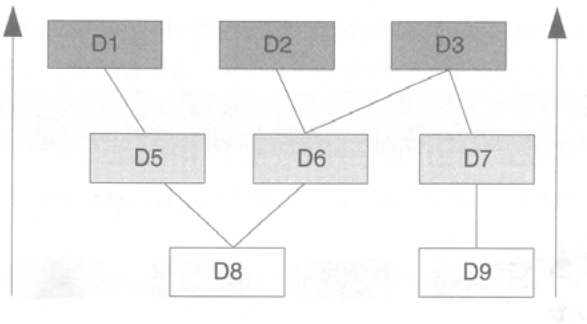
Top-Down: *Schrittweise* Integration *ausgehend* von den *Business Cases*

- Vorteile: Ausführbares Produkt-Framework früh verfügbar, Prototypen für Demos, Framework für Testfälle.
- Nachteile: Zusätzlicher (hoher) Aufwand für Test-Stubs, Integration von Hardware erfolgt erst spät (zusätzliches Risiko).



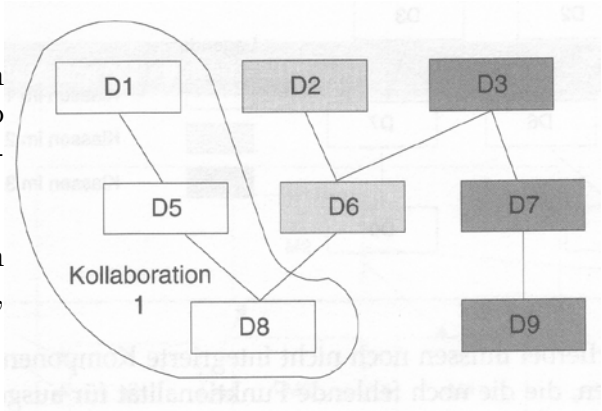
Bottom-Up: *Schrittweise* Integration *ausgehend* von der *Hardware (Low-Level)*

- Vorteile: Stabiles System, Schrittweise Integration Richtung Business Cases.
- Nachteile: Ausführbares Gesamtsystem ist spät verfügbar, Zusätzlicher Aufwand für Prototypen und Test Drivers



Build: *Schrittweise* Integration *entsprechend* den *Business Cases* über alle Layer hinweg. Phasen-orientierte Integration.

- Vorteile: Frühe Verfügbarkeit von funktionalen Anforderungen, Prototyp und Demo, Berücksichtigung priorisierter Anforderungen möglich.
- Nachteile: Wiederverwendung von Komponenten kann schwierig sein, Regressionstests erforderlich.



Diskutieren Sie die Begriffe Validierung und Verifikation

Validierung: Erwartung des Kunden vs. Umsetzung („Wurde das richtige Produkt entwickelt?“) Beispiel: Akzeptanz- und Abnahmetests (Prüfung gegen Anforderungen).

Verifikation: Spezifikation vs. Umsetzung („Wurde das Produkt richtig entwickelt?“) Beispiel: Komponententests (Prüfung gegen die technische Spezifikation)

Blackbox/Whitebox Testing (Skizze)

Black Box Tests

- Anforderungen/Spezifikation als Grundlage
- Unabhängig von der Realisierung der Module.
- Data-drive (Input/Output).
- Anforderungsüberdeckung.
- Äquivalenzklassenbildung.
- Keine genaue Fehlerortung möglich.

White Box Tests

- Sourcecode als Grundlage.
- Wissen über internen Aufbau notwendig.
- Logic-driven.
- Kontrollflussüberdeckung.
- Äquivalenzklassen von internen Verzweigungen und Schleifen.
- Ermöglicht Fehlererkennung und –lokalisierung.

Was sind Äquivalenzklassen?

Hierbei werden die Eingabedaten in Klassen mit dem selben (äquivalenten) Verhalten eingeteilt und anschließend Repräsentant jeder Klasse für einen Testfall gewählt. Testfälle müssen gültige (Normalfall, Sonderfall) UND ungültige Eingabewerte (Fehlerfall) berücksichtigen.
Beispiel: int quersumme(int i)

Äquivalenzklasse	***Repräsentant***	***Gültig?***
[0,...,MAX_INT]	123	Ja
[MIN_INT,...,0[	-1	Nein
Fließkommazahl	1.5	Nein
Keine Zahl	A	Nein

Was ist eine Grenzwertanalyse?

Grenzwertanalyse ist ein Spezialfall einer Äquivalenzklasse, bei welcher man die Repräsentanten um die Klassen-Grenzen wählt, da hier besonders leicht Fehler auftreten.

Beschreiben Sie den Prozess des Testfallbestimmens. Was muss dokumentiert werden? (2019-07-04)

Eine Möglichkeit Testfälle zu bestimmen ist die Äquivalenzklassenzerlegung. Die Testfälle (und Ergebnisse) müssen ausführlich dokumentiert werden als Information für Entwickler, für die Wiederholbarkeit und Berichterstattung und um sie als Kommunikati-

onswerkzeug einsetzen zu können. Die Dokumentation sollte mindestens folgende Informationen enthalten:

- **Typ:** Normal-, Spezial- oder Fehlerfall?
- **Vorbedingung:** Eingabedaten, Zustand des System vor Testausführung.
- **Beschreibung:** Was soll der Test zeigen?
- **Äquivalenzklasse:** Falls eine Äquivalenzklassenzerlegung vorgenommen wurde: Auf welcher Klasse basiert der Testfall?
- **Erwartetest Ergebnis:** Welches Ergebnis/Systemzustand wird nach der Ausführung des Tests erwartet. Der Test ist erfolgreich wenn der tatsächliche Zustand mit dem Erwarteten übereinstimmt.
- **Tatsächliches Ergebnis:** Ergebnis/Systemzustand nach dem Test.

Nennen und beschreiben Sie unterschiedliche Sichten auf die Wartung

- **Activity-View:** Änderung des Software Systems nach Auslieferung (Delivery) und Inbetriebnahme (Deployment bzw. Product Launch).
- **Process-View:** Schritte zur Durchführung einer Wartungsaufgabe.
- **Phase-Oriented-View:** Die Wartungsphase beginnt mit der Auslieferung und Inbetriebnahme und Ende mit "Stilllegung" des Softwareproduktes.

Wartungskategorien

	Correction	*Enhancement*
Proactive	**Preventive** - Verbesserung im Hinblick auf zukünftige Wartung (z.B. Ergänzung der Dokumentation)	**Perfective** - Produktverbesserung (Erweiterungen, Verbesserung der Effizienz).
Reactive	**Corrective** - Bug- und Fehlerkorrektur	**Adaptive** - Berücksichtigung geänderter externer Anforderungen (Hardware, Softwareänderungen)

Nennen Sie drei Techniken zur Wartung

- **Herstellen des Produktverständnisses:** Das Verständnis „fremder“ Codestücke kann – speziell bei mangelnden Aufzeichnungen – sehr zeitaufwendig sein. Key-Tools sind Code-Browsers und essentiell ist eine klare und präzise Dokumentation.
- **Reengineering:** Überprüfung und Überarbeitung des Software Codes. Stellt eine gravierende und teure Form der Änderung/dar.
- **Reverse Engineering:** Analyse der Software im Hinblick auf Komponenten und deren Zusammenhänge. Dabei hilft es, auf Basis des Software Codes, Modelle auf einem höheren Abstraktionsniveau zu erstellen z.B. UML-Modelle aus Code zu generieren.

Welche Phasen umfasst der Wartungsprozess?

Wartungsprozesse beinhalten dieselben Phasen wie der Life-Cycle Prozess; einen speziellen Stellenwert nehmen die Anforderungsänderungen ein. Siehe Abbildung 4.1.

Block 5

Ausgewählte Software Prozesse

Planbasierte Methoden

Systematische Prozesse sind durch ihre Struktur plan-driven und orientieren sich eheran Abläufen mit Schwerpunkt auf Produkten und Dokumentation. Beispiele: Wasserfallmodell, V-Modell (XT), RUP, Inkrementelle Modelle.

Erklären und skizzieren Sie das Wasserfall Modell. Nennen Sie auch Vor- und Nachteile.

Das Wasserfall-Modell ist eine Umsetzung des Life-Cycles aus den 80er Jahren das einfach Anzuwenden ist und eine Schwerpunkt auf Dokumentation legt. Der Ablauf ist skizziert in Abbildung 5.1. Zu den Vorteilen zählt:

- Backtracking zu früheren Entwicklungsphasen.
- Risikominimierung durch „Abschluss“ einer Phase.
- Weite Verbreitung und hoher Bekanntheitsgrad.
- Strikte Trennung der einzelnen Phasen.
- Unterstützung von kleinen Entwicklungsteams.

Nachteile:

- Alle Tasks einer Phase müssen abgeschlossen werden (keine parallele Entwicklung möglich).
- Starke Auswirkung von Fehlern in früheren Phasen auf das Entwicklungsprojekt.

Anwendung bei guten Kenntnissen über die Anforderungsdomäne und nur bei klar definierten und vollständigen Anforderungen.

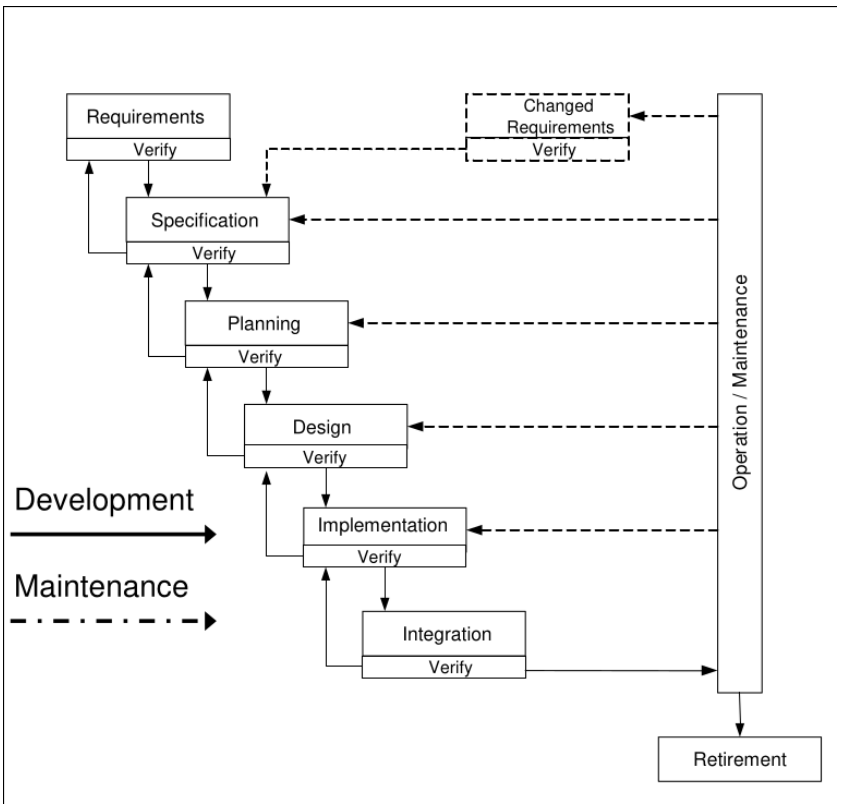


Abbildung 5.1: Ablauf von Projekten mit dem Wasserfall-Modell.

Erklären und skizzieren Sie das V-Modell. Nennen Sie auch Vor- und Nachteile.

Das V-Modell ist ein Vorgehensmodell welches dem Wasserfallmodell ähnlich ist, aber zusätzliche Phasen für die Qualitätssicherung einführt, welche den Entwicklungsphasen gegenüberstehen. Der Ablauf ist in Abbildung 5.2 dargestellt.

Vorteile:

- Spezifikationsphase vs. Realisierung und Testen.
- Kontext von Produkten und Tests.
- Verschiedene Abstraktionslevels (User, Architektur, Implementierung).
- Fehlerbehandlung in frühen Phasen (Reviews).
- Basiskonzept für VM 97 und VM XT.

Nachteile:

- Klare Beschreibung der Systemanforderungen ist wichtig.
- Hoher Dokumentationsaufwand.

- Kritisch bei unklaren/veränderlichen Anforderungen.

Anwendung bei großen Projekten im öffentlichen Bereich mit klar definierten Anforderungen.

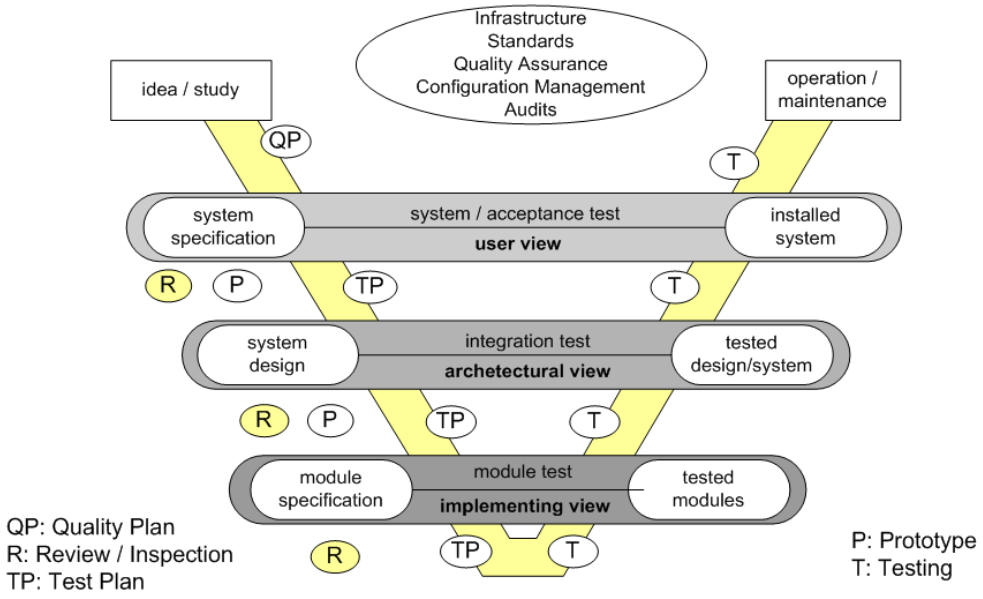


Abbildung 5.2: Ablauf des V-Modell

V-Modell XT

Rational Unified Process, RUP

- Iterative und inkrementeller Workflow.
- Integriertes Anforderungsmanagement.
- Komponenten-orientierte Architektur.
- Modellierung durch das UML Methodenframework.
- Produkt-Verifikation an Meilensteinen.
- Änderungsmanagement (supporting discipline).

Vorteile:

- Real-world Szenarien.
- Werkzeugunterstützung
- Vordefinierte Liste mit erforderlichen Artefakten.

Nachteile:

- Hohe Komplexität.
- Hoher Dokumentationsaufwand.
- Anbieterabhängigkeit

Anwendungsbereich: Grosse Projekte durch eine ganzheitliche Prozess-Sicht auf das gesamte Projekt (inkl. Deployment).

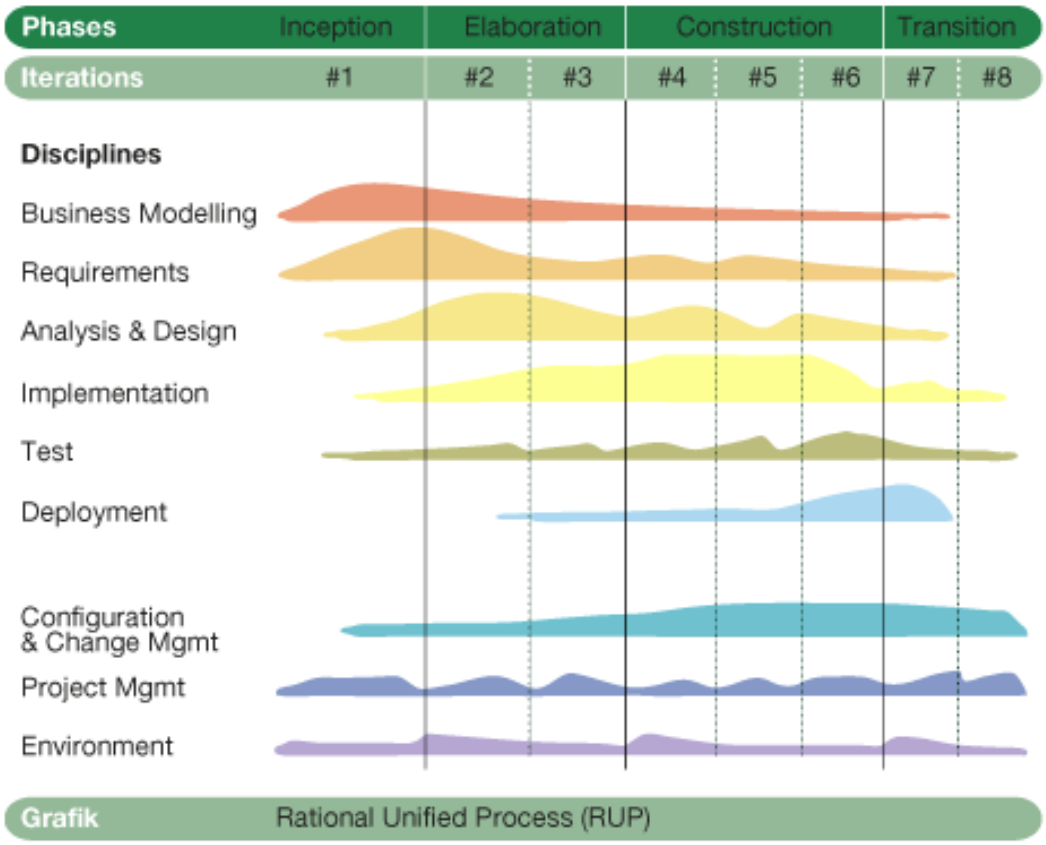


Abbildung 5.3: Ablauf des RUP-Prozesses.

Was gehört zu den 12 Prinzipien des Agilen Manifests? Multiple-Choice (2019-07-04)

12 agile Prinzipien:

1. Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.

2. Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.

3. Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.

4. Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.

5. Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.

6. Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.

7. Funktionierende Software ist das wichtigste Fortschrittsmaß.

8. Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.

9. Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.

10. Einfachheit – die Kunst, die Menge nicht getaner Arbeit zu maximieren – ist essenziell.

11. Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.

12. In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

Erklären und skizzieren Sie den SCRUM Prozess. Nennen Sie auch Vor- und Nachteile.

SCRUM ist ein agiler Software Prozess für kleine aber hoch-effiziente Teams (bei großen Projekten sind auch mehrere Teams möglich). Das flexible Prozessmodell erlaubt es auf sich ändernde Anforderungen im Projektablauf zu reagieren. Entsprechend dem Agilen Manifest stehen (Teil-)Produkte frühzeitig zur Verfügung (frühe und kontinuierliche Auslieferung). Generell erfüllt SCRUM einen hohen Anteil der agilen Prinzipien.

Der SCRUM-Prozess ist dargestellt in Abbildung 5.4.

Vorteile:

- Wenige Regeln, leicht verständlich.
- Hohe Effektivität durch Selbstorganisation.
- Adaptiv/Tolerant gegenüber Anforderungsänderungen und neu Priorisierung.

- Hohe Transparenz durch regelmäßige Meetings und Backlogs.
- Geringer Administrations- und Dokumentationsaufwand.

Nachteile:

- Kein Gesamtüberblick über die komplette Projektstrecke.
- Hoher Kommunikations- und Abstimmungsaufwand.
- Erschwerte Koordination mehrerer Entwicklungsteam bei Großprojekten.
- Potentielle Verunsicherungen aufgrund fehlender Zuständigkeiten und Hierarchien.
- Potentielle Unvereinbarkeit mit bestehenden Unternehmensstrukturen.

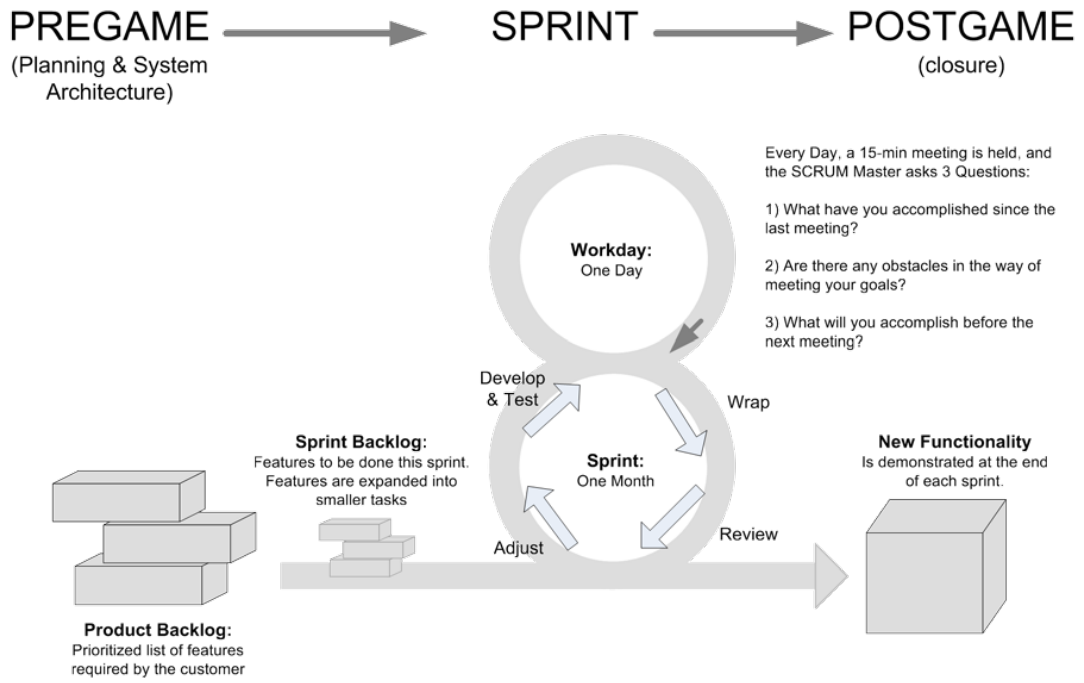


Abbildung 5.4: Ablauf des SCRUM-Prozesses.

Block 6

Modellierung von Anwendungsprozessen

Block 7

Gastvortrag: Feedback

Nennen Sie fünf Arten wie man Feedback im Rahmen eines Software-Projektes einholen kann (2019-07-04)

In Software-Projekten kann Feedback auf viele Arten eingeholt werden:

- Prototypen
- Code Review, Pair Programming
- Neue Features sofort Testen (am besten durch Kunden)
- Usability Tests
- Architektur top-down aus dem Elfenbeinturm (???)

Warum ist Feedback wichtig und wie soll es aussehen? (2019-03-20)

Menschen verstehen die Umwelt durch Interaktion und Rückmeldung, d.h. Feedback ist für das Verständnis der Umwelt sehr wichtig. Zusätzlich kann durch rechtzeitiges Feedback die Zeit in sinnlosen Projekten minimiert und Fehler früh erkannt werden.

Beim Feedback geben ist zu beachten:

- Höflich und wertschätzend
- Negatives unter vier Augen
- Feedback ist *eigene Meinung* – nicht die „Wahrheit“
- Über das Problem, nicht die Person sprechen: „*Der* Code war *für mich* schwer zu lesen“ vs. „*Dein* Code *ist* schwer zu lesen“.
- Alternativen anbieten
- Nicht erst bei Problemen, d.h. regelmäßig positive Rückmeldungen, Feedback institutionalisieren (Pair-Programming, Code-Reviews, 1:1 Meetings, Pull Requests)

Andere Prüfungen mit ähnlicher/selber Frage:

- 2019-01-21

Block 8

QS in SE-Projekten

Qualitätssicherung

Qualitätssicherung (QS) besteht in der Durchführung von Verifikation und Validierung in jeder Phase der Software-Herstellung

Welche Qualitätskriterien gibt es in der Softwareentwicklung?

- Testbare Anforderungen
- Verfolgbare Entwicklung der Anforderungen (mit Modellen) in testbare Produkte.

Was ist ein Software Review und wozu dient er?

Ein Review ist ein formal geplanter und strukturierter Analyse- undBewertungsprozess, in dem Projektergebnisse einem Team von Gutachtern präsentiert und von diesen kommentiert oder genehmigt werden. Reviews dienen vor allem zur qualitativen Beurteilung von Produkten und Prozessen,die quantitativ nur schwer oder gar nicht beurteilt werden können (z.B. Modelle, Dokumente)

8.0.1 Welche Arten von Reviews kennen Sie? Erläutern Sie die wesentlichen Aspekte der einzelnen Reviewarten und geben Sie jeweils ein konkretes Beispiel an.

Inspektion: Formale Prüfung gewöhnlich durch gleichgestellte Personen nach vorgegebenen Regeln und Checklisten. Primärziel –> Fehlerzustände finden.

Technisches Review:

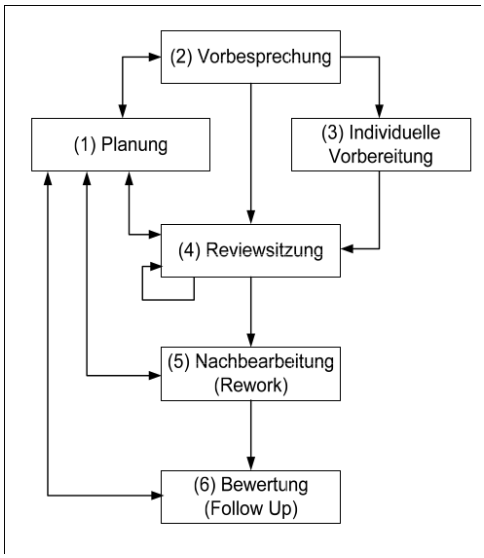
Welche Rollen finden sich typischerweise bei Reviews und welche Aufgaben nehmen die unterschiedlichen Rollen wahr?

- **Moderator** – Leiter des Reviews

- **Leser**
- **Gutachter** – Kommentierung des Reviewobjektes
- **Schreiber** – Protokollschreiber verfasst Protokoll (Notizen während der Reviewsitzung)
- **Autor** – Klärung von offenen Fragen. Keine Kommentierung und Rechtfertigung der Lösungen.

Beschreiben und skizzieren Sie den Ablauf eines Reviews. (2019-01-21)

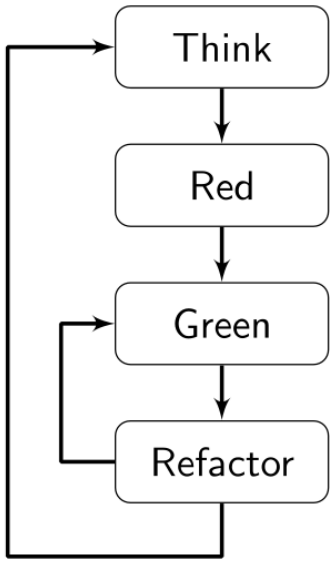
1. **Planung:** Objekt, Prüfziele, Auslösekriterien (Einstiegskritierien), Teilnehmer, Ort und Zeit festlegen.
2. **Vorbesprechung:** Vorstellung des Prüfobjekts bei komplexen/neuen Produkten.
3. **Individuelle Vorbereitung:** Intensive Einzeldurcharbeitung.
4. **Review-Sitzung:** Eigentliche Durchführung des Reviews. Gemeinsames Lesen, Aufzeichnung von Mängeln (entdecken, nicht korrigieren!).
5. **Nachbearbeitung:** Die dokumentierten Mängel korrigieren.
6. **Bewertung:** Die Korrekturen überprüfen.



Test Driven Development

Test Driven Development ist ein Entwicklungs- und Designparadigma für Software, bei dem das Testen von Programmkomponenten dazu verwendet wird, den gesamten Prozess der Softwareentwicklung zu leiten. Test Driven Development ist eine Designstrategie, die das Testen vor dem Erstellen des Quellcodes ansiedelt und mit Bezug auf die Abläufe vorrangig behandelt. Das Ziel liegt darin, die Qualität der Software maßgeblich zu erhöhen und den Wartungsaufwand im Nachhinein zu verringern. TDD wird meist im Rahmen agiler Methoden und insbesondere beim Extreme Programming verwendet.

- **Think:** Spezifikation des Tests
- **Red:** Implementierung des Tests – Test schlägt fehl!
- **Green:** Implementierung der zutestenden Klasse oder Komponent. – Test ist erfolgreich!
- **Refactor:** Veränderung einer Implementation -> Funktionalität bleibt erhalten -> Test sollte nie wieder fehl schlagen! – Lernen aus Test, Veränderung des Prozess



Was bringt TDD in Verbindung mit Continuous Integration?

Die Zeit für viele Unit Tests ist wohl investiert da sie die Integrations-Tests wesentlich vereinfachen. (bottom-up testing)

Block 9

Design Patterns

3 Creational Patterns beschreiben.

- **Singleton** – Provision of a single instance only
- **Factory** – Method in a derived class creates associates
- **Prototype** – Factory for cloning new instances from a prototypical instance

4 Structural Patterns beschreiben.

- **Facade** – Facade simplifies the interface for a subsystem
- **Adapter** – Translator adapts a server interface for a client
- **Proxy** – One object approximates another
- **Bridge** – Abstraction for binding one of many implementations

Block 10

PM - Teil 2

Zeichnen und erklären Sie die Motivationspyramide nach Maslow. Nennen Sie auch ein Alternativ Modell für menschliche Motivation.

Die Motivationspyramide nach Maslow ist eine polythematische Theorie zur Erklärung menschlichen Verhaltens welche von fünf hierarchischen Kategorien menschlicher Bedürfnisse ausgeht. Erst wenn die unteren Stufen in einem gewissen Ausmaß befriedigt sind wird die darauf Aufbauende verfolgt. Die Pyramide ist in Abbildung 10.1 dargestellt und umfasst folgende Kategorien:

1. **Grund- oder Existenzbedürfnisse (Physiologische):** Lebensnotwendige Bedürfnisse wie z.B. Nahrung und Schlaf.

2. **Sicherheit:** Absicherung bezüglich Schutz des Lebens, des Eigentums, Lebensstandard, Schutz vor Arbeitslosigkeit, Unfallfolgen bis hin zur Altersvorsorge.

3. **Sozialbedürfnis:** Gruppenzugehörigkeit, soziale Akzeptanz, Freundschaft, Zuneigung, etc.

4. **Anerkennung und Wertschätzung (Achtung):** Unterscheidung zwischen Selbst- und Fremdachtung. Wird gesteuert durch das Erleben der eigenen Leistung und der resultierenden Anerkennung. Umfasst auch Kompetenz, Status, Respekt.

5. **Selbstverwirklichung:** Das Verlangen des Individuums, zu verwirklichen, was es potentiell ist, d.h. seine potentiellen Fähigkeiten zu entfalten. Menschen in dieser Stufe werden weitgehend durch die Freude motiviert und versuchen einen starken Einfluss auf ihre Umwelt auszuüben. Sie sind kreativ aktiv, leistungsorientiert und versuchen ihre Fähigkeiten weiterzuentwickeln.

   Ab dieser Stufe kommt es zu einer gewissen Eigendynamik: je stärker dieses Bedürfnis befriedigt wird, desto bestimmender wird es für das Verhalten.

Alternative monothematische Theorien sind z.B. der Sexualtrieb (Freud) oder Minderwertigkeitskomplex (Adler). Andere polythematische Theorien sind z.B. das ERG-Modell (Existence, Relatedness, Growth) nach Adlerfer oder die zweidimensionale Arbeitszufriedenheitstheorie von Herzberg.

Abbildung 10.1: Bedürfnispyramide nach Maslow. Von LMU Dozent Medizin (Diskussion) 04:47, 23. Sep. 2017 (CEST) - Eigenes Werk (Originaltext: selbst erstellt), CC BY-SA 3.0 de, `https://commons.wikimedia.org/w/index.php?curid=62696203`

Nennen Sie je zwei Faktoren die Einfluss/keinen Einfluss auf die Produktivität von Menschen haben

Fanktoren ohne Einfluss:

- Programmiersprache
- Berufserfahrung
- Anzahl der Fehler
- Gehalt

Fanktoren mit Einfluss:

- Teampartner
- Arbeitsplatz
- Anzahl der Fehler
- Gehalt

Nennen und erklären Sie sechs Wege, die Teambildung verhindern

- Defensives Management - jede Entscheidung wird vom Projektmanagement getroffen, die Teammitglieder werden entmündigt
- Bürokratie - das sinnlose Produzieren von Projektdokumenten, das die Teammitglieder von ihrer eigentlichen Arbeit abhält
- Physikalische Trennung - die räumliche Trennung von Personen, die eigentlich zusammen arbeiten sollen
- Zersplitterung der Zeit der Mitarbeiter - die Aufteilung der Mitarbeiter auf mehrere Projekte
- Qualitätsreduktion der Produkte - die Qualitätsreduktion erfolgt unter dem Argument der Kostenreduktion; der Effekt ist eine geringere Identifikation der Mitarbeiter mit dem Projekt
- Scheintermine - unrealistische Termine bewirken nur, dass sich die Mitarbeiter nicht ernst genommen fühlen und daher nur eine geringe Identifikation mit dem Projekt aufbringen werden
- Cliquenkontrolle - Veränderung der Teamstruktur während des Projektes bzw. Zerschlagung von Teams nach dem Projektende

Nennen Sie vier Formen und vier Richtlinien zur Team-Organisation. (2019-07-04)

Formen der Team-Organisation:

- klassisch hierarchische Organisation
- typische Matrix-Organisation
- „Chef-Programmierer“-Team
- offen strukturiertes Team
- SWAT-Team
- XP-Team (Extreme Programming)

Richtlinien zur Team-Organisation:

- Setzen Sie bessere Leute ein
- Setzen Sie weniger Leute ein

- Stellen Sie den Teammitgliedern Aufgaben, die zu ihren Fähigkeiten passen.
- Stellen Sie den Teammitgliedern Aufgaben, die zu ihren Interessen passen.
- Achten Sie mittelfristig auf die persönliche Entwicklung der Mitarbeiter.
- Achten Sie auf eine balancierte und harmonische Mischung.
- Lösen Sie sich von Leuten, die nicht zum Team passen.