

Advanced Software Engineering 1 - HS20
Pascal Brunner - brunnpa7

Inhaltsverzeichnis

| | |
|--|----------|
| 1 Einführung ins Software Engineering | 3 |
| 1.1 Was ist Software Engineering | 3 |
| 1.1.1 Was ist Engineering? | 4 |
| 1.1.2 Ziele und Mittel | 4 |
| 1.1.3 5 P im Software Engineering | 4 |
| 1.2 Software Entwicklung als Prozess | 5 |
| 1.2.1 Application Lifecycle Management | 5 |
| 1.3 Qualitätsaspekte | 5 |
| 1.4 SWEBOK | 5 |
| 2 Software Engineering Prozesse | 6 |
| 2.1 Software-Prozessmodelle | 6 |
| 2.1.1 Begrifflichkeiten | 6 |
| 2.1.2 Klassifizierung von Software-Entwicklungsprobleme | 6 |
| 2.1.3 Aspekte des Softwareentwicklungsprozess | 7 |
| 2.2 Tailoring des Prozesses für ein Projekt | 7 |
| 3 Requirements Engineering | 8 |
| 3.1 RE agilen Software-Prozessmodellen | 8 |
| 3.2 Einführung und Grundlagen | 8 |
| 3.2.1 Gründe und Symptome für mangelhaftes RE | 8 |
| 3.2.2 Risiken bei nicht korrektem Erfassen von Requirements | 8 |
| 3.2.3 Haupttätigkeiten | 9 |
| 3.2.4 Einflussfaktoren | 9 |
| 3.2.5 Kommunikation | 10 |
| 3.2.6 Was sind wichtige Anforderungen an einen Requirements Engineer | 10 |
| 3.2.7 die 3 Arten von Anforderungen | 11 |
| 3.2.8 Qualitätsanforderungen | 11 |
| 3.2.9 Wechselwirkung zwischen Was und Wie | 11 |
| 3.2.10 wrap-up | 11 |
| 3.3 System und Systemkontext abgrenzen | 12 |
| 3.3.1 Systemgrenze und Umwelt | 12 |
| 3.3.2 Systemkontext | 13 |
| 3.4 System- und Kontextgrenzen bestimmen | 13 |
| 3.4.1 Systemgrenze festlegen | 14 |
| 3.4.2 Systemkontext dokumentieren | 14 |
| 3.5 Anforderungen ermitteln | 15 |
| 3.5.1 Anforderungsquellen | 15 |
| 3.5.2 Kano-Model | 16 |
| 3.5.3 Ermittlungstechniken | 16 |
| 3.6 Anforderungen dokumentieren | 20 |
| 3.6.1 vier Arten von Dokumentation | 20 |

| | | |
|----------|--|-----------|
| 3.6.2 | Dokumentengestaltung | 21 |
| 3.6.3 | Arten der Dokumentation | 21 |
| 3.6.4 | Dokumentenstrukturen | 21 |
| 3.6.5 | Verwendung von Anforderungsdokumenten | 21 |
| 3.6.6 | Qualitätskriterien für das Anforderungsdokument | 22 |
| 3.7 | Anforderungen natürlich sprachlich dokumentieren | 22 |
| 3.7.1 | Sprachliche Effekte | 22 |
| 3.7.2 | Konstruktion von Anforderungen mittels Satzschablone | 23 |
| 3.8 | Anforderungen modellbasiert dokumentieren | 24 |
| 3.8.1 | Der Modellbegriff | 24 |
| 3.8.2 | Zielmodelle | 25 |
| 3.8.3 | Use Cases | 26 |
| 3.8.4 | Drei Perspektiven auf die Anforderungen | 26 |
| 3.9 | Dokumentation im agilen Umfeld | 27 |
| 3.9.1 | User Stories | 28 |
| 3.9.2 | Technical Stories | 28 |
| 3.9.3 | Definition of Done (DoD) | 28 |
| 3.9.4 | Story Maps | 29 |
| 3.10 | Anforderungen prüfen und abstimmen | 29 |
| 3.10.1 | Grundlage | 29 |
| 3.10.2 | Qualitätsaspekte | 30 |
| 3.10.3 | Prinzipien der Prüfung von Anforderungen | 31 |
| 3.10.4 | Techniken zur Prüfung von Anforderungen | 31 |
| 3.10.5 | Abstimmung von Anforderungen | 33 |
| 3.11 | Anforderungen verwalten | 34 |
| 3.11.1 | Attribute von Anforderungen | 34 |
| 3.11.2 | Sichten auf Anforderungen | 35 |
| 3.11.3 | Priorisierung von Anforderungen | 35 |
| 3.11.4 | Verfolgbarkeit von Anforderungen | 35 |
| 3.11.5 | Versionierung von Anforderungen | 36 |
| 3.11.6 | Verwaltung von Anforderungsänderungen | 37 |
| 3.12 | Werkzeugunterstützung | 38 |
| 3.12.1 | Allgemeine Werkzeugunterstützung | 38 |
| 3.12.2 | Modellierungswerzeuge | 38 |
| 3.12.3 | Requirements-Management-Werkzeuge | 39 |
| 3.12.4 | Werkzeugeinführung | 39 |
| 3.12.5 | Beurteilung von Werkzeugen | 40 |
| 4 | Software Architektur | 41 |
| 4.1 | Grundlagen | 41 |
| 4.1.1 | Definitionen | 41 |
| 4.1.2 | Grundlegende Konzepte | 42 |
| 4.1.3 | Bausteine | 42 |
| 4.1.4 | Schnittstellen | 43 |
| 4.1.5 | Softwarearchitekturbeschreibung | 43 |
| 4.1.6 | Qualität und Nutzen der Softwarearchitektur | 45 |
| 4.1.7 | Softwarearchitekturentwurf | 46 |
| 4.1.8 | Der Softwarearchitekt | 47 |
| 4.2 | Entwurf von Softwarearchitekturen | 47 |
| 4.2.1 | Entwurfsprinzipien | 47 |
| 4.2.2 | Architekturzentrierte Entwicklungsansätze | 47 |
| 4.2.3 | Architekturmuster | 47 |
| 4.2.4 | Entwurfsmuster | 47 |

Kapitel 1

Einführung ins Software Engineering

1.1 Was ist Software Engineering

Softwarereprodukte sind:

- Allgemein: entwickelt für den Verkauf an Anwender
- Massgeschneidet: für spezielle Anforderungen / Bedürfnisse

Für ein Softwareprodukt gibt es einige Aspekte die dazugehören:

- Entwurfsdokumentation
- Lastenhaft
- Systemprototypen
- Systementwurf
- Testumgebung → rund 50 % des Aufwandes fällt ins Testing
- Konfigurationsmanagement
- Systembeschreibung
- Produktsupport
- Bedienungshandbücher

Software ist gemäss IEEE 610.12 wie folgt definiert: *Die Programme, Verfahren, zugehörige Dokumentation und Daten, die mit dem Betrieb eines Rechnersystems zu tun haben*

Dabei ist speziell, dass man die Software nicht anfassen kann, das erschwert bspw. das Erkennen von Fehler

- Fehler beobachtbar nur
 - in den Wirkungen beim Ablauf auf Rechner
 - indirekt über die Dokumentation der Software
- Kein Materialwert
- keine physikalische Grenzen
- Fehler sind schwierig erkennbar
- Entwicklungsstand und Qualität schwer zu beurteilen
- Scheinbar leicht zu ändern

Wozu dient die Software

- Löst ein Problem
- Wenn das Problem komplex ist, ist auch die Software komplex
- Fachdomäne muss verständlich sein ⇒ Bindeglied ist oftmals ein Business Analyst
- Software kann die Realität verändern

1.1.1 Was ist Engineering?

- Übersetzt heisst es Ingenieurswissenschaft

1.1.2 Ziele und Mittel

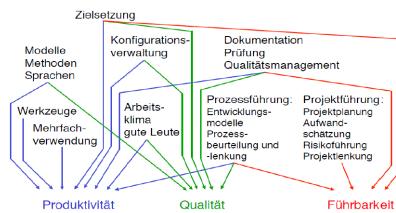


Abbildung 1.1: Ziele und Mittel der SE

1.1.3 5 P im Software Engineering

- Projekte
 - klein, gross, sehr gross
 - Forschungsprojekt, Wartungsprojekt
 - Budget, Zeit, Risiko
- Personen
 - Rollen (BA, Architekten, Entwickler, Tester)
- Prozesse
 - Vorgehensmodelle (Phasen, Aktivitäten, Vorlagen)
 - Wasserfall, Unified Process, Hermes, Agile
- Produkte und Leistungen
 - Artefakte (Zwischenprodukte, Endprodukte)
 - Qualität (Messung, Metriken)
- Paradigmen
 - Strukturierte Analyse, Objekt-orientiert, Service-Oriented-Architecture (SOA), Patterns

1.2 Software Entwicklung als Prozess

Es gibt einen Produktlebenszyklus, welcher bei der ersten Idee startet und bei der Ausmusterung der Softwarelösung endet.

Ein Prozess hilft dabei, dass man schlussendlich zu dem Endprodukt kommt, welcher man entsprechend auch möchte. Ein Softwareprozess ist dann fertig, wenn die Software ausgemustert wurde und **NICHT** wenn die Software live geht.

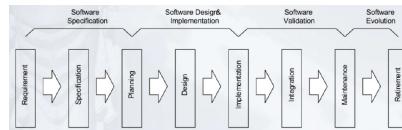


Abbildung 1.2: Übersicht über den Software Lifecycle Prozess

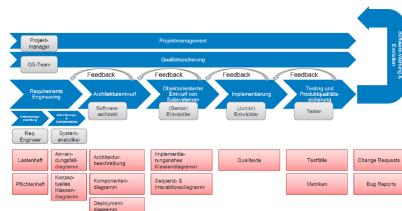


Abbildung 1.3: Software Engineering Aktivitäten, Rollen und Artefakten

1.2.1 Application Lifecycle Management

Ist in direkte Bereichen aufgeteilt:

- Governance (Steuerung)
- Development (Entwicklung)
- Operations (Betrieb)

Zusammengehörend ist hierbei auch DevOps (Development - Operations), welches zur Koppelung von Entwicklung und Betrieb führt

1.3 Qualitätsaspekte

für die Produktqualität ist mit ISO 25010 klar definiert. Dabei gibt es interne und externe Qualitäten



Abbildung 1.4: Qualität gemäß ISO 25010

⇒ Um die Qualität zu testen, braucht man klare Metriken / Messwerte → Hierbei hilft dann beispielsweise das automatische Testen

1.4 SWEBOK

Kapitel 2

Software Engineering Prozesse

Ein strukturierter Prozess drängt sich für die Software-Entwicklung auf

2.1 Software-Prozessmodelle

2.1.1 Begrifflichkeiten

Prozess: Ablauf eines Vorhabens mit der Beschreibung der Schritte (Aktivitäten), der beteiligten Personen (Rollen), der für diesen Ablauf benötigten Informationen und der dabei entstehenden Informationen (Artefakte)

- Entwicklung und Wartung von Software sind Prozesse
- Ein (Software)

2.1.2 Klassifizierung von Software-Entwicklungsprobleme

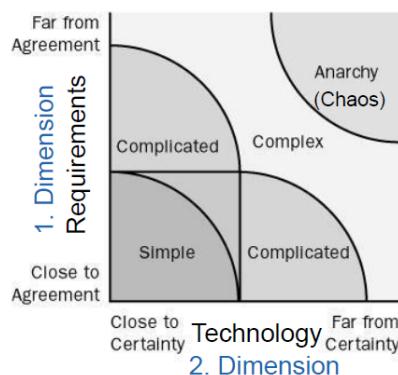


Abbildung 2.1: Einordnung von Probleme

- heutzutage gibt es keine simple-Projects mehr
- häufig sind die Requirements sehr komplex und häufig auch die Technology → endet häufig im *complicated* oder *complex* Part
- Dazu sollte eigentlich noch eine 3. Dimension gehören → Die Menschen (Skills, Intelligence Level, Experience, Attitudes, Prejudices)

2.1.3 Aspekte des Softwareentwicklungsprozess

- Ermitteln und Analyse der Anforderungen
- Architektur / Entwurf (Design)
- Implementierung / Umsetzung (Construction)
- Test (Testing)
- Inbetriebnahme (Deployment, Configuration, Start of Operation)
- Wartung / Betrieb (Maintenance / Operation)

2.2 Tailoring des Prozesses für ein Projekt

Kapitel 3

Requirements Engineering

3.1 RE agilen Software-Prozessmodellen

Aus klassischer Sicht, fixiert man die Requirements und schätzt die Zeit, welche man dazu verwendet. Die neue Methode kehrt dieses Dreieck um, man fixiert die Ressourcen und Zeit und schätzt die Requirements

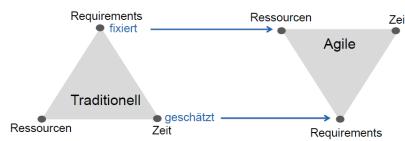


Abbildung 3.1: Vergleich von RE von früher zu heute

Vorteil: Der Kunde weiss noch nicht im Detail, welche Funktionalitäten man erhält, jedoch weiss man wie viel man entsprechend zahlt.

3.2 Einführung und Grundlagen

Warum scheitern Projekte?

- Unklare Anforderungen und Ziele
- Schlechte Kommunikation
- Mangelhaftes Stakeholdermanagement

→ Je später man einen Fehler findet, desto teurer wird dieser Fehler zu reparieren

3.2.1 Gründe und Symptome für mangelhaftes RE

- Kommunikationsprobleme
- Ergebnisorientierung → zu wenig Wert auf Diskussion
- Selbstverständlichkeit → vieles wird angenommen und wird nicht explizit genannt
- Projektdruck → Auftraggeber erwarten kurzfristige Ergebnisse

3.2.2 Risiken bei nicht korrektem Erfassen von Requirements

- Fehlende Anforderungen
- ungenaue und falsche interpretierte Anforderungen
- Unechte Anforderungen

- implizite Anforderungen
- widersprüchliche Anforderungen
- Schleichende Änderung der Anforderungen

Anforderung

IEEE-Norm: *Eine Bedingung oder Fähigkeit, die von einem Benutzer (Person oder System) zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird*

Stakeholder

Ein Stakeholder eines Systems ist

- eine Person, Personen gruppe oder eine Organisation
- die direkt oder indirekt
- Einfluss auf die Anforderungen des betrachteten Systems hat

Requirements Engineering

RE ist ein systematischer und disziplinierter Ansatz zur Spezifikation und zum Management von Anforderungen mit folgenden Zielen:

1. Die relevanten Anforderungen zu kennne, Konsens unter den Stakeholder über die Anforderungen herstellen, die Anforderungen konform zu vorgegebenen Standards zu dokumentieren und die Anforderungen systematisch zu managen
2. Die Wünsche und Bedürfnisse der Stakeholder zu verstehen, zu dokumentieren sowie die Anforderungen zu spezifizieren und zu managen, um das Risiko

3.2.3 Haupttätigkeiten

- Ermitteln
 - Anforderungen der Stakeholder zu gewinnen, zu detaillieren und zu verfeinern
- dokumentieren
 - Anforderungen adäquat beschreiben
- Prüfen und abstimmen
 - Erfüllen der Qualitätskriterien für Anforderungen prüfen
- verwalten
 - Anforderungen strukturieren
 - für unterschiedliche Rollen aufbereiten
 - konsistent zu ändern und umzusetzen

3.2.4 Einflussfaktoren

Dabei spielt die Domäne und die Umgebung einen wichtige Rolle, diese gelten als Einflussfaktoren.

- Anwendungstyp
- Anwendungsdomäne
- Räumliche Verteilung

Des Weiteren ist die Organisation und Prozesse relevant:

- Prozesse (Scrum, RUP, Hermes etc.)
- Best Practices
- Normen und Standards (IEEE, ISO, DIN)

3.2.5 Kommunikation

Es gibt verschiedene Arten der Kommunikation (schriftlich, mündlich), dabei ist bei dieser Kommunikation wichtig, dass man eine gemeinsame Sprache entwickelt.

Dadurch kann man sicherstellen, dass die Fehlerquellen möglichst minimiert werden. Denn der Sender und Empfänger codieren bzw. decodieren jeweils die Nachrichten.

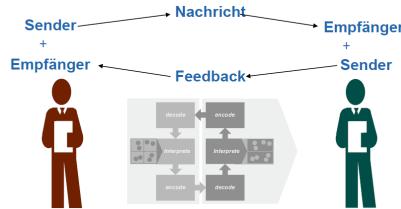


Abbildung 3.2: Modell des Sender und Empfängers

Die zwischenmenschliche Kommunikation bewegt sich immer auf zwei Ebenen:

1. Inhaltsebene

rationaler, sachlicher Austausch von Informationen

2. Beziehungsebene

Emotionen, bewusst und unbewusste Wahrnehmungen und Gefühle

Es sind vielmehr häufig Probleme auf der Beziehungsebene, die eine rationale, sachliche Kommunikation zwischen vers. Personen oder Gruppen erschweren

3.2.6 Was sind wichtige Anforderungen an einen Requirements Engineer

- Analytisches Denken
- Empathie
- Kommunikationsfähigkeit
- Konfliktlösungsfähigkeit
- Moderationsfähigkeit
- Selbstbewusstsein
- Überzeugungsfähigkeit

Requirements Engineer vs. Product Owner

RE:

- Anforderungsermittlung
- Anforderungsdokumentation bzw. Wissensvermittlung
- Anforderungsvalidierung
- Anforderungsmanagement

PO:

- Sicherstellen, dass das Entwicklungsteam konstanten betriebswirtschaftlichen Wert liefert
 - Managen aller Stakeholder
 - Kontinuierliche Versorgung des Entwicklungsteams mit den hochrangigen Einträgen aus dem Backlog
- ⇒ Die Rolle des PO ist breiter gefächert als ein herkömmlicher RE

3.2.7 die 3 Arten von Anforderungen

- Funktionale Anforderung
- Qualitätsanforderungen → Grosser Einfluss auf die Systemarchitektur
- Randbedingungen → Schränken den Lösungsansatz ein

→ Qualitätsanforderungen und Randbedingungen werden oftmals als *Nicht-funktionale Anforderungen* zusammengefasst.

3.2.8 Qualitätsanforderungen

Qualitätsanforderungen müssen explizit dokumentiert werden

3.2.9 Wechselwirkung zwischen Was und Wie

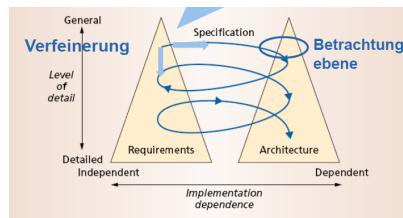


Abbildung 3.3: Wechselwirkung zwischen Was und Wie

Granularität von Anforderungen

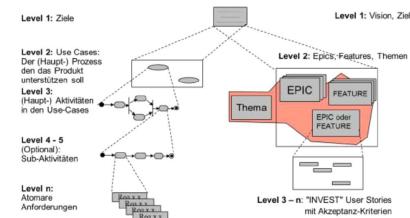


Abbildung 3.4: Granularität von Anforderungen

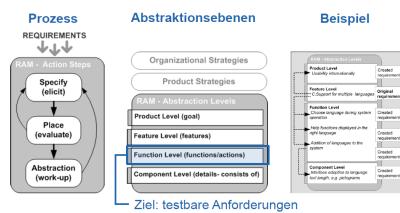


Abbildung 3.5: Detaillierungsebene

3.2.10 wrap-up

- Gutes RE ist wichtig, da viele Fehler schon in dieser Phase entstehen
- Symptome für mangelhaftes RE sind fehlende und unklare Anforderungen
- Vier Haupttätigkeiten (Ermitteln, Dokumentieren, Prüfen, Verwalten)
- Sprache ist das wichtigste Mittel der Kommunikation

- wichtige Eigenschaften (analytisches Denken, Empathie, Konfliktlösungsfähigkeit, Moderationsfähigkeit, Selbstbewusstsein, Überzeugungsfähigkeit)
- drei Arten von Anforderungen (funktionale Anforderungen, Qualitätsanforderungen, Randbedingungen)

3.3 System und Systemkontext abgrenzen

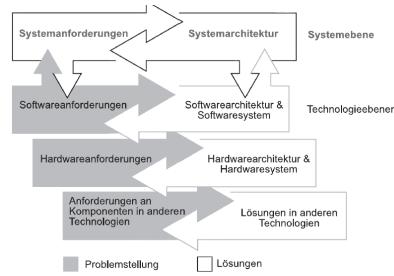


Abbildung 3.6: Uebersicht der Systembegriffe

Systembegriff:

- Unterscheidung von System
 - Ein System ist ein dynamisches Ganzes
 - bestimmte eigenschaften und Verhaltensweise
 - besteht aus Teilen die miteinander verknüpft sind
 - das Verhalten des Ganzen beeinflusst wird ...
 - Problemstellung und Lösung sollte klar getrennt sein
 - Problemstellung ist meist länger stabil, als die Lösung
 - Zu einer Problemstellung findet man normalerweise mehr als eine mögliche Lösung
- ⇒ Aus diesen Gründen streben wir die getrennte Formulierung von Problemstellung und Lösung auf jeder Ebene der Entwicklung an

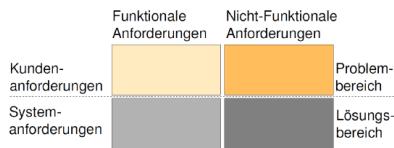


Abbildung 3.7: Formulierung Problemstellung

3.3.1 Systemgrenze und Umwelt

Die Systemgrenze speziiert das geplante System von seiner Umgebung

- gestaltbaren und veränderbarer Teil der Realität von Aspekten in der Umgebung
- Es ist zu achten, dass Elemente innerhalb des Systems eine höhrere Verbindungsichte

Systembetrachtung

1. Wirkung nach aussen:

- Man betrachtet das System als Blackbox
- Fokus auf Input und Ausput

2. Strukturorientiert:

- Strukturorientierte Systembetrachtung
- Man bricht das System auf und definiert aus welchen Elementen das System besteht

3.3.2 Systemkontext

Definition: Der Systemkontext ist der Teil der Umgebung eines Systems, der für die Definition und das Verständnis der Anforderungen des betrachteten Systems relevant ist

- Identifikation der materiellen und immateriellen Aspekten, die eine Beziehungen zum System haben
- Der für die Anforderung des System relevante Ausschnitt der Realität wird als Systemkontext bezeichnet

Aspekte

tbd

Konsequenzen eines fehlerhaften oder unvollständigen Kontextes

- Unvollständige oder fehlerhafte Systemkontakte führen zu fehlerhaften Anforderungen
- Entwicklung wird aufgrund von unvollständigen oder fehlerhaften Anforderungen durchgeführt
- Diese Fehler bleiben bei der Überprüfung, ob das System die spezifizierten Anforderung abdeckt, unentdeckt. Fehler treten erst später im Betrieb auf.

Systemkontext und Anforderungskontext

- Ursprung der Anforderung eines Systems liegt im Systemkontext
- Eine Anforderung ist in einem spezifischen Kontext definiert
- Je vollständiger dieser Kontext einer Anforderung definiert ist, umso geringer ist die Wahrscheinlichkeit der Fehlinterpretation

3.4 System- und Kontextgrenzen bestimmen

- **Systemabgrenzung:** Welche Aspekte werden durch das geplante System abgedeckt
- **Kontextabgrenzung:** Grenze des Kontext zur irrelevanten Umgebung

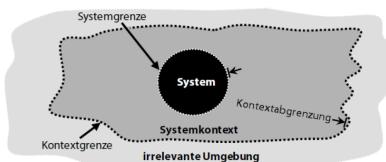


Abbildung 3.8: System- und Kontextabgrenzung

3.4.1 Systemgrenze festlegen

Definition Systemgrenze: Die Systemgrenze separiert das geplante System von seiner Umgebung. Sie grenzt den im Rahmen des Entwicklungsprozesses gestaltbaren und veränderbaren Teil von Aspekten der Umgebung ab, die durch den Entwicklungsprozess nicht verändert werden können.

- Die Systemgrenze grenzt den Konstruktionsgegenstand zur Umgebung hin ab
- Durch die Wahl der Systemgrenze wird festgelegt, welche Aspekte das zu konstruierende System (scope) abdeckt und was ausserhalb des Systems liegt
- Sämtliche Aspekte die innerhalb der Systemgrenze liegen, können somit verändert bzw. gestaltet werden

Grauzone

- Die Systemgrenze ist oft erst gegen Ende des RE Prozesses präzise festgelegt
- Zu Beginn sind bspw. einige oder mehrere Schnittstellen nur Unvollständig bekannt
- Die Grauzone kann während dem Prozess laufend verschoben werden

Kontextgrenze bestimmen

Definition: Die Kontextgrenze separiert den relevanten Teil der Umgebung eines geplanten Systems vom irrelevanten Teil, d.h. dem Teil der Umgebung, der keinen Einfluss auf das geplante System und damit auch keinen Einfluss auf die Anforderungen dieses Systems hat. BSc I Modul ASE1 LE 03 - RE Kap. 2 System und Systemkontext abgrenzen

- Die Kontextgrenze differenziert in der Umgebung des geplanten Systems zwischen Kontextaspekte und Aspekte die nicht relevant sind
- diese werden idealerweise explizit niedergeschrieben, dass eine impliziten Annahmen getroffen werden können

3.4.2 Systemkontext dokumentieren

Diese Dokumentation kann auf unterschiedliche Arten dokumentiert werden

- Datenflussdiagrammen
- Use Case Diagramme

→ Ist eine sehr einfache Art und Weise mit den Stakeholdern die Grenzen zu definieren

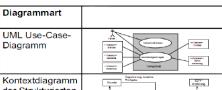
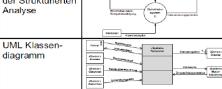
| Art der Kontextbildung | Schnittstelle | Diagrammart | |
|------------------------|----------------|--|---|
| logisch | unspezifiziert | UML Use-Case-Diagramm |  |
| | Ein-/Ausgaben | Kontrollflussdiagramm der Strukturierten Analyse |  |
| | Ein-/Ausgaben | UML Klassendiagramm |  |
| | Nachrichten | UML Sequenzdiagramm |  |
| physisch | Kanäle | UML Verteilungsdiagramm |  |

Abbildung 3.9: Arten der Dokumentation von Systemkontext

3.5 Anforderungen ermitteln

3.5.1 Anforderungsquellen

- Stakeholder
 - Person oder Organisation, die Einfluss auf die Anforderungen hat
- Dokumente
 - enthalten wichtige Informationen, aus denen Anforderungen gewonnen werden können
- Altsysteme
 - Systeme im Betrieb oder Konkurrenzsysteme → ergeben Basisanforderungen

Stakeholder

- Identifikation der Stakeholder ist zentrale Aufgabe des RE
- sind wichtige Quellen für Anforderungen
- RE hat Aufgabe die oft widersprechenden Anforderungen und Ziele der Stakeholder in Einklang zu bringen



Abbildung 3.10: Ansicht von versch. Stakeholder

Dokumentation Die Anforderungsquellen sollte hinsichtlich der Stakeholder dokumentiert werden

- normalerweise Funktion
- weitere Personen- und Kontaktdata
- zeitliche und räumliche Verfügbarkeit während Projekt
- Relevanz
- Wissensgebiet und -umfang
- Ziele und Interessen

Vereinbarungen

- Manchmal lohnt es sich die Aufgaben, Verantwortungsbereiche und Weisungsbefugnisse festzulegen
- Dabei geht man auf die Rechte und Pflichten ein
- Dies beugt Mängel an Motivation bzw. Konflikte vor → Stakeholder sollen sich als Projektbeteiligte sehen

3.5.2 Kano-Model

Ist für die Anforderungskategorisierung vorgesehen

- Basisfaktoren → wird als selbstverständlich und absolut notwendig (unbewusste Anforderungen) betrachtet
Führt zur massiven Unzufriedenheit
Bei Erfüllung gibt es keine positive Stimmung (wird erwartet)
Beobachtungstechniken und dokumentenzentrierte Techniken sind guten Ermittlungstechniken
- Leistungsfaktoren → Sind explizit gewünschte (bewusste) Anforderungen. Der User kann auf einige Leistungsfaktoren verzichten, jedoch nicht auf zu viele
Steigert Zufriedenheit
gut durch Befragungstechniken zu ermitteln
- Begeisterungsfaktoren → wird erst durch die Benutzung ersichtlich und begeistert die Nutzer zusätzlich unbewusste Anforderungen - werden nicht erwartet
durch Kreativitätstechniken gut zu ermitteln

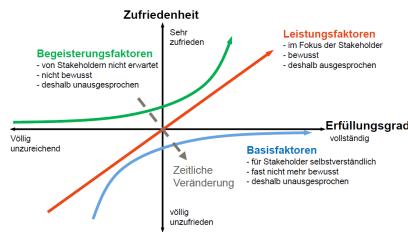


Abbildung 3.11: Ansicht der verschiedenen Anforderungskategorisierung

⇒ Mit der Zeit wird ein Begeisterungsfaktor zu einem Leistungsfaktor und anschliessend zu einem Basisfaktor

3.5.3 Ermittlungstechniken

Ermittlungstechniken erfüllen den Zweck, die bewussten, unbewussten und unterbewussten Anforderungen der Stakeholder herauszufinden.

- Ergebung ist zeit- und arbeitsintensiv
- ein geeigneter Satz an Techniken muss pro Projekt aufeinander abgestimmt werden
- Jede Erhebung benötigt mehrere Runden
- Befragungstechniken
 - Interview
 - Fragebogen
- Kreativitätstechniken
 - Brainstroming
 - Brainstroming paradox
- dokumentenzentrierte Techniken
- Beobachtungstechniken
- unterstützende Techniken

⇒ Die Technikwahl ist massgebend für den Erfolg

| Technik | Anforderungskategorien | | |
|------------------------|------------------------|-------------------|-----------------------|
| | Basisfaktoren | Leistungsfaktoren | Begeisterungsfaktoren |
| Interviews | + | ++ | + |
| Fragebogen | - | + | - |
| Kreativitäts-Techniken | - | ++ | ++ |
| Beobachtung | ++ | + | + |
| Dokumenten-zentriert | ++ | + | - |

Abbildung 3.12: Zusammenhang Ermittlungstechniken mit dem Kano-Modell

- Unterscheidung je nach Anforderungen
- Termin- und Budgetvorgaben
- Verfügbarkeit der Stakeholder
- Erfahrung des RE
- Chancen und Risiken des REs

Risikofaktoren

- Menschliche Einflüsse
 - gute Kommunikation, Anforderungsart, Detaillierungsebene, Erfahrung
 - kognitive Fähigkeit der Stakeholder
- Organisatorische Einflüsse
 - Festpreis oder Werkvertrag

...

Befragungstechniken

- Aussage des Stakeholders über die Anforderungen an das System aus seiner Perspektive
- Stakeholder muss Wissen explizit ausdrücken
- tendenziell durch den RE durchgeführt

Interview

- mehrere vorgegebene Fragen werden gestellt und protokolliert
- Fragen werden im Gespräch geklärt
- durch geschickte Fragen können unbewusste Anforderungen ermitteln

Vorteile

- sehr direkt und gezielt auf den Sachverhalt eingehen
- individuell auf Gesprächspartner anpassbar
- Wenige Missverständnisse

Nachteile

- Aufwändige Vorbereitung
- Qualität vom Interviewer abhängig

- Auswertung ist arbeitsintensiv

Fragebogen

- offene und oder geschlossene Fragen
- gut online möglich
- in kurzer Zeit viele Personen befragen

Vorteile

- einfache und schnelle Auswertung
- Anonymität möglich
- Schriftliche Aussagen können nur schwer dementiert werden
- einen Schlag viele Stakeholder

Nachteile

- nur abgefragt, was der RE bereits kennt oder vermutet
- Angst vor schriftlicher Befragung
- Einfluss von Drittpersonen möglich
- Mangelhaftes Ausfüllen hat mühsame Rückfragen zur Folge
- persönlicher Kontakt fehlt
- Motivation der Befragten kann geringer sein als bei einem persönlichen Gespräch

Kreativitätstechniken

- Entwicklung von innovativen Anforderungen (Visionen)
- nicht für die Ermittlung von detaillierten Anforderungen

Brainstorming

- 5 - 10 Personen in vorgegebener Zeit Ideen gesammelt
- Erst im Anschluss werden diese geprüft und analysiert

Vorteil

- finden von innovativen Ideen und ausgefallenen Lösungen
- kann gut in Sackgass-Situationen verwendet werden
- Einfache Durchführung
- geringe Kosten
- Ausnutzung von Synergieeffekten infolge Gruppenbildung

Nachteil

- Stark abhängig der Teilnehmer
- Oftmals sehr viel unbrauchbar
- Gefahr in Abschweifung

- Aufwändige Selektion der Ideen
- Gefahr von gruppodynamische Konflikte

Brainstorming Paradox

- genau gleich wie Brainstorming
- Ergebnisse werden gesammelt, welche nicht erreicht werden sollen

Perspektivenwechsel

- Arbeitet mit Perspektivenwechsel
- bspw. Sechs-hut-Denken (jeder Hut nimmt andere Perspektive ein (neue Idee, Zweifel etc.))
- ganzheitliche Betrachtung / animiert neue Sichtweisen einzunehmen
- nicht geeignet für Detailarbeiten und Präzision / Akzeptanz der Technik

Dokumentenzentrierte Techniken

- verwenden Lösungen und Ideen von bestehenden Systemen weiter
- ...

Systemarchäologie

- Untersuchender Funktionalitäten eines bestehenden Systems

Perspektivenbasiertes Lesen und Wiederverwendung

- Perspektivenbasiertes Lesen
- Wiederverwendung

Beobachtungstechniken

- RE beobachtet Abläufe und hinterfragt diese
- RE hat die Chance ineffiziente Prozesse zu erkennen

Feldbeobachtung

- RE ist vor Ort
- Dokumentiert unmittelbar vor Ort
- Evtl. Audio oder Videoaufzeichnung

Apprenticing

- RE geht in die Lehre
- Arbeitet aktiv mit
- UNklare Handlungen werden sofort hinterfragt
- Machtverhältnis Stakeholder und RE ist umgedreht

Unterstützende Techniken

- Mindmapping
- Workshops
- Use Cases Modellierung
- User stories
- Prototypen

3.6 Anforderungen dokumentieren

Einflussfaktoren für das Dokumentieren

- Kritikalität
- Risiko
- Wichtigkeit der Kunden
- Rechtliche Verbindlichkeit
- Grad der Standardisierung
- Erfahrung

3.6.1 vier Arten von Dokumentation

- Dokumentation für gesetzliche Zwecke

Prinzip: muss aus den Gesetzen und Standards abgeleitet werden
ist untrennbarer Bestandteil des Produkts

- Dokumentation zum Zwecke der Bewahrung

Prinzip: Das Team entscheidet, was zum Zwecke der Bewahrung dokumentiert wird

- Dokumentation für Kommunikationszwecke

Prinzip: zusätzliches Kommunikationsmittel
Verlief die Kommunikation erfolgreich, sollte das Dokument archiviert werden

- Dokumentation zur Zwecke von Überlegungen

Prinzip: nachdenkende Person entscheidet über die Dokumentform
muss ihre Wahl nicht rechtfertigen
Dokument kann verworfen werden

⇒ Ist kein Selbstzweck, sondern soll die Kommunikation zwischen Stakeholdern erleichtern, vor allem zwischen dem Anfordernden und dem Entwicklungsteam

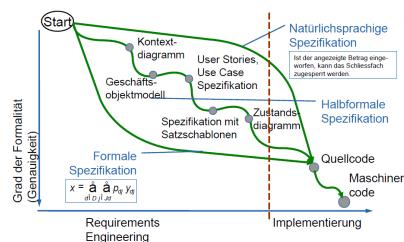


Abbildung 3.13: Varianten der Dokumentationsformen

3.6.2 Dokumentengestaltung

tbd

3.6.3 Arten der Dokumentation

tbd

3.6.4 Dokumentenstrukturen

Standard oder individuelle Dokumentenstruktur

Vorteil von Standards:

- Einarbeitungszeit kleiner
- Schnellere Erfassung von ausgewählter Inhaltsebene
- ...

Was gibt es für Standards?

- RUP - Rational Unified Process (IBM)
Enthält untersch. Artefakten
- Standard ISO/IEC/IEEE 29148:2011
- V-Modell
Lastenheft
Pflichtenheft

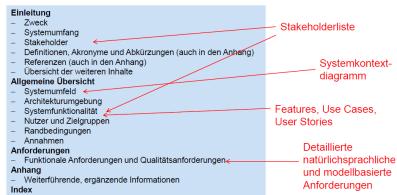


Abbildung 3.14: Angepasste Standardinhalte SRS

3.6.5 Verwendung von Anforderungsdokumenten

- Planung
- Architekturentwurf
- Implementierung
- Test
- Änderungsmanagement
- Systemnutzung und Systemwartung
- Vertragsmanagement

3.6.6 Qualitätskriterien für das Anforderungsdokument

- Eindeutigkeit und Konsistenz
- Klare Struktur
- Modifizierbarkeit und Erweiterbarkeit
- Vollständig
- Verfolgbarkeit

3.7 Anforderungen natürlich sprachlich dokumentieren

3.7.1 Sprachliche Effekte

- Realität → Tiefenstruktur (Interpretation)
- Oberflächenstruktur → Tiefenstruktur (Nachfragen)
- Durch die sprachlichen Effekte können gleiche Dinge unterschiedliche Interpretation erfolgen
- die natürliche Sprache ist mehrdeutig und interpretierbar
- bei den Vorgängen Wahrnehmung und Darstellung treten so genannte Transformationsprozesse auf

Es gibt fünf für das RE relevantesten Transformationsprozesse:

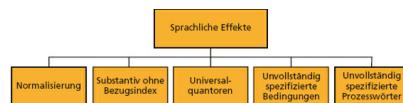


Abbildung 3.15: die wichtigsten Transformationsprozesse

Normalisierung

- Ähnliche Begriffe in einem Verb zusammenfassen
- Durch diese Umwandlung gehen wichtige Informationen verloren
- Bei der sprachlichen Analyse werden alle Normalisierungen identifiziert
- Schwierigkeit ist, dass bei der Normalisierung jeweils komplexe Prozesse darunterstecken können
bspw. Rückgabe → Rückgabe muss genauer analysiert werden

Substantive ohne Bezugsindex

- Substantive bergen ähnliche Prozesswörter die Gefahr der unvollständigen Spezifizierung → ohne ausreichende Bezugsindizes
bspw. der Anwender, der Daten, das System
besser: der angemeldete Anwender

Universalquantoren

- Universalquantoren sind Angaben über die Häufigkeit
- Hinterfragen ob das für alle Objekte gilt
bspw: nie, immer, alle, irgendeiner

unvollständig spezifizierte Bedingungen

- Unvollständige spezifizierte Bedingungen
- Eintritt der Bedingung wird beschrieben (if-Fall), aber nicht der Else-Fall

Unvollständig spezifizierte Prozesswörter

- Prozesswörter brauchen mehr als ein Substantiv, um vollständig spezifiziert zu sein
bspw. übertragen: was? von wo? wohin?

3.7.2 Konstruktion von Anforderungen mittels Satzschablone

- Ein einfach erlern- und einsetzbarer Ansatz zur Reduzierung sprachlicher Effekte während der Formulierung von Anforderung ist die Satzschablonen
- Die Satzschablonen unterstützt den Autor einer Anforderung effektiv dabei, qualitativ
- ... vervollständigen...

Schritt 1

Das Festlegen der Verbindlichkeit durch die Verben, 'muss', 'sollte', 'wird' kann im Text der Anforderung erfolgen. Ändern sich die Verbindlichkeiten, dann ändern sich auch die Anforderungen

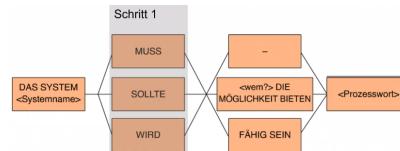


Abbildung 3.16: Schritt eins der Satzschablonen

Schritt 2

Geforderte Funktionalität, drucken, speicher, übertragen ist ein Prozess und wird durch Verben beschrieben → Gefordertes Systemverhalten wird in Schritt 2 beschrieben

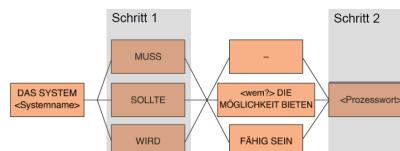


Abbildung 3.17: Schritt zwei der Satzschablonen

Schritt 3

- Selbstständige Systemaktivität
Das System führt den Prozess selbstständig durch
- Benutzerinteraktion
Das System stellt dem Nutzer die Prozessfunktionalität zur Verfügung
- Schnittstellenanforderungen
Das System führt einen Prozess in Abhängigkeit von einem Dritten (bspw. Fremdsystem) aus, ist an sich passiv und wartet auf ein externes Ereignis

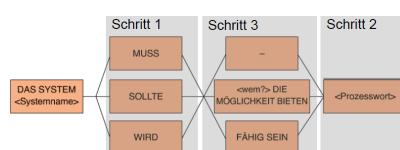


Abbildung 3.18: Schritt drei der Satzschablonen

Schritt 4

Manche Prozesswörter benötigen zur Ergänzung ein oder mehrere Objekte → das Prozesswort *drucken* wird um was oder wo ergänzt

Schritt 5

Für Anforderungen ist es typisch, dass die geforderte Funktionalität nicht fortwährend, sondern nur unter bestimmten zeitlichen oder logischen Bedingungen ausgeführt oder zur Verfügung gestellt wird

- zeitliche Bedingung (sobald, nachdem)
- Logische Bedingung (Falls)
- Wenn sollte vermieden werden, weil es zeitlich oder logisch eingesetzt werden kann



Abbildung 3.19: vollständige Satzschablone

3.8 Anforderungen modellbasiert dokumentieren

3.8.1 Der Modellbegriff

Ein Modell ist ein abstrahierendes Abbild einer existierenden Realität oder Vorbild für eine zu schaffende Realität

- Modelle können nicht falsch oder richtig sein, sondern vielmehr kann passend bzw. nicht-passend für ein System sein
- ...

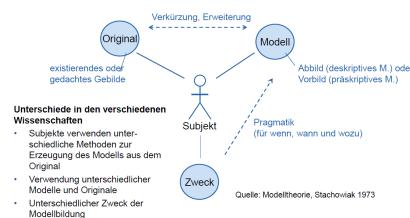


Abbildung 3.20: Eigenschaften von Modellen

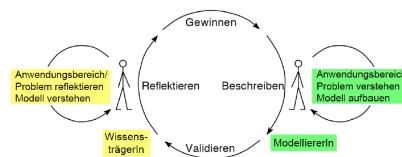


Abbildung 3.21: Prinzipschema von Modellen

- Modellbildung ist ein iterativer Prozess
- Bedeutet auch immer Refelktieren über das Original
- Ist auch ein Verstehens- und Konsensbildungsprozess

Konzeptuelle Modellierungssprachen

- Konzeptuelle Modelle, die Anforderungen eines Systems dokumentieren, werden als Anforderungsmodelle bezeichnet

wird oftmals UML eingesetzt

UML besteht aus einer Menge von zum Teil komplementären Modellierungssprachen, die speziell im RE eingesetzt werden um die Anforderungen eines Systems aus verschiedenen Perspektiven zu modellieren

Unterschied konzeptuelle Modelle in der Systementwicklung der Verwendung von konzeptuellen Modellen in der Anforderungsdokumentation

Systementwicklung: es werden Lösungsaspekte dokumentiert

RE: es werden Anforderungen aus verschiedenen Perspektiven modelliert

- Zur Modellierung Konzeptueller Modelle werden Konzeptuelle Modellierungssprachen eingesetzt, die über deren Syntax (Modellierungselemente und deren gültige Kombination) und Semantik (Bedeutung der Modellierungselemente) definiert werden
- Konzeptuelle Modelle können hinsichtlich ihres Formalisierungsgrades in informale, semiformale und formale Modellierungssprache eingeteilt werden
- Der Formalisierungsgrad ist abhängig vom Umfang, in dem Syntax und Semantik der Sprache formal definiert sind

Anforderungsmodelle

- Konzeptuelle Modelle, die Anforderungen eines Systems dokumentieren, werden als Anforderungsmodelle bez

3.8.2 Zielmodelle

- Ziele → Und-Oder-Graphen

- Szenarien

Personas und Szenarien

Storyboards

Use Case Diagramme

- Funktionsperspektive

UML-Aktivitätsdiagramme

UML-Sequenzdiagramme

Datenflussdiagramme

- Strukturperspektive

UML-Klassendiagramme

Entity-Relationship-Diagramme

- Verhaltensperspektive

UML-Zustandsdiagramme

Statecharts

3.8.3 Use Cases

Use Cases sind noch nicht alle Anforderungen, sondern werden durch weitere Anforderungen mit UI-Sketches bzw. Prosa definiert

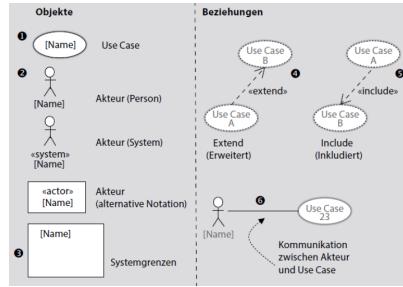


Abbildung 3.22: Modellelemente Use Case

Use Case Spezifikation

- ergänzen die überblicksartigen Use-Case-Diagramme durch eine genaue Spezifikation der wesentlichen Eigenschaften einzelner Use Cases
- Hierzu wird in der Regel für jeden relevanten Use Case separat eine vorgegebene Schablone ausgefüllt

3.8.4 Drei Perspektiven auf die Anforderungen

Anforderungsmodellierung in der Strukturperspektive

Mit Entity-Relationship-Diagramm (ERD) → in der Regel wird jedoch die UML Notation verwendet

Häufiger wird mit UML-Klassendiagramme gearbeitet

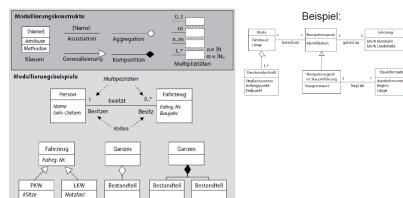


Abbildung 3.23: Elemente des UML Klassendiagramm

Anforderungsmodellierung in der Funktionsperspektive

Datenflussdiagramme werden fast nicht mehr verwendet, da veraltet → wurde durch UML-Aktivitätsdiagramme abgelöst

Das UML-Aktivitätsdiagramm:

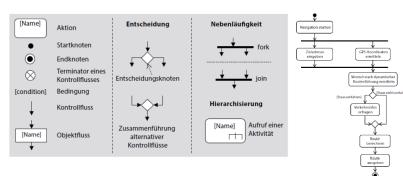


Abbildung 3.24: Elemente des UML Aktivitätsdiagramm

⇒ sehr gut geeignet für das Modellieren von Parallel-Abläufen

Anforderungsmodellierung in der Verhaltensperspektive

Statechart ist ebenfalls veraltet → wird durch UML-Zustandsdiagramme abgelöst

UML-Zustandsdiagramme:

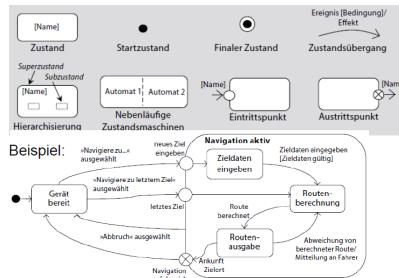


Abbildung 3.25: Elemente des UML Zustandsdiagramm

3.9 Dokumentation im agilen Umfeld

Recap: Wie funktioniert Scrum

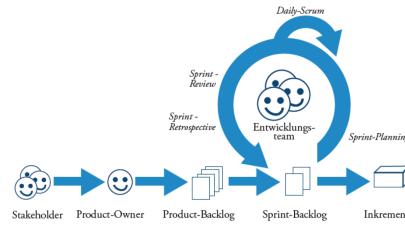


Abbildung 3.26: Überblick Vorgehen nach Scrum

- Normalerweise werden in einem Sprint alle Tätigkeiten zur Entwicklung (inkl. Analyse) des Inkrements durchgeführt, um die erforderlichen Informationen genau zu dem Zeitpunkt zu erzeugen, zu dem sie auch benötigt werden
- Je nach Domäne und Projektrandbedingungen muss aber auch schon vorher und nachher noch RE betrieben werden
- Product Owner und Entwicklungsteam benötigen Tätigkeiten aus verschiedenen RE-Disziplinen für ihren Aufgabenbereich
- Die Auswahl der einzelnen Techniken in diesen Tätigkeiten kann sich zum Teil in einem Projekt immer wieder ändern, da sie sehr stark von den aktuellen Gegebenheiten abhängen
- Die Verantwortlichkeit für die Auswahl dem Entwicklungsteam zu überlassen, kommt diesem Umstand sehr entgegen, da sie am besten die aktuell vorliegende Situation einschätzen und so die optimale Auswahl treffen können
- Für jeden Typ von Informationen, der zur Dokumentation ansteht, sind folgende Fragen zu stellen
 - Wie lange nach der Entstehung wird die Information benötigt?
 - Wann soll die Information dokumentiert werden?
 - Wie soll dokumentiert werden?

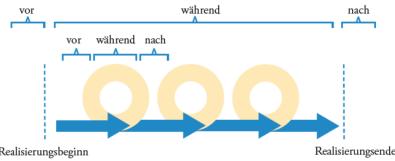


Abbildung 3.27: Integrations-Optionen von RE in Scrum

3.9.1 User Stories

- Eine User Story beschreibt eine Funktionalität, die für den Kunden oder Benutzer eines Produkts oder Systems von Wert ist
- Sie besteht aus der schriftlichen Beschreibung der Funktionalität, Gesprächen über die Funktionalität und Akzeptanzkriterien, die Details vermitteln und festlegen, wann eine User Story vollständig umgesetzt ist
- ⇒ eine User Story ist in einem Sprint durchzuführen, entsprechend muss man diese 'abschneiden'
- Akzeptanzkriterien legen fest, unter welchen Bedingungen ein Product-Backlog-Eintrag (bspw. User Story) als umgesetzt gilt und erfolgreich abgenommen wird

3.9.2 Technical Stories

- Beschreiben keine Funktionalität → werden oft als Technical Stories bezeichnet
- umfassen technische oder andere nicht-funktionalite Aspekte, die nicht über die Implementierung einzelner User Stories abgedeckt werden (bspw. Refactoring)

3.9.3 Definition of Done (DoD)

→ man eignet sich im Team *Wann ist man fertig*

- Die Definition of Done bezieht sich immer auf das Ergebnis eines Entwicklungszyklus
- Kann bei Bedarf angepasst werden
- Der Scrum-Guide definiert die DoD als ein Artefakt, das bei allen Projektbeteiligten ein gemeinsames Verständnis dafür erzeugt, welche formalen Kriterien erfüllt sein müssen, damit die Arbeit an einem System- oder Produktinkrement als abgeschlossen gilt

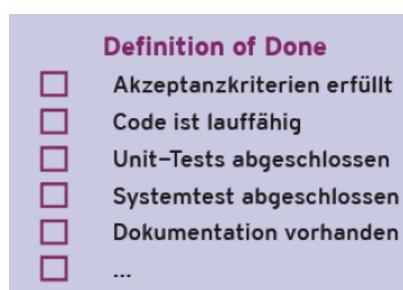
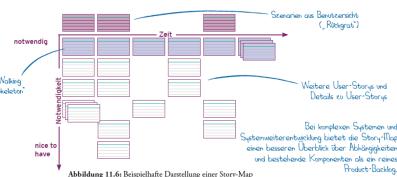


Abbildung 3.28: Beispiel DoD

3.9.4 Story Maps

- Use Cases und User Stories haben eine ähnliche Ausrichtung:
 - bilden einzelne Systemfunktionalitäten aus der Sicht eines Benutzers ab
 - liefern Übersicht über das System
- Use Cases sind jedoch überlicherweise auf größerer Ebene als User Stories
- Ein Use Case umfasst mehrere oder sogar viele User Stories
- Epic wiederum kann in seinem Umfang einem groben Use Case entsprechen
- Dokumentationstechniken von Use Cases können auch auf User Stories übertragen werden
- User Stories vs. Use Cases
 - Es werden nur jene Anforderungen im Detail ausgearbeitet, die in kommenden Iterationen realisiert werden
→ User Stories unterstützen diesen Prozess
 - Funktionen welche erst später geplant sind, werden zunächst nur grob aufgenommen
 - User Stories alleine eignen sich hingegen nicht besonders gut dazu, das Wissen über Anforderungen langfristig aufzubewahren, sofern dies notwendig und angebracht ist
 - Die Dokumentation mittels Use Cases, das Festhalten von Entscheidungen, Szenarien mit ergänzenden Kontextinformationen usw. sind dazu besser geeignet



3.10 Anforderungen prüfen und abstimmen

Die Zielsetzung der Prüfung von Anforderungen besteht darin, Anforderungen dahingehend zu überprüfen, ob sie festgelegten Qualitätskriterien (bspw. Korrektheit oder Vollständigkeit) genügen, um etwaige Fehler in den Anforderungen möglichst frühzeitig im RE erkennen und beheben zu können.

3.10.1 Grundlage

- Ziel: Auffinden von Fehlern → bspw. Mehrdeutigkeit, Unverständlichkeit, Widersprüche
- Anforderungsdokumente sind die Grundlagen
- Fehler beeinträchtigen alle weiteren Entwicklungstätigkeiten
- Fehlerfortpflanzung: nicht nur die Anforderung muss angepasst werden, sondern allenfalls auch Architektur, Artefakten etc.

- Freigabe von Anforderungen
- Verträge zwischen Auftraggeber und Auftragnehmen basieren oft auch auf Anforderungsdokumenten

ABstimmung von Anforderungen

- Ziel der Abstimmung von Anforderungen ist es, unter den relevanten Stakeholder ein gemeinsames und übereinstimmendes Verständnis hinsichtlich der Anforderungen an das zu entwickelnde System zu erarbeiten
- Widersprechende Anforderungen erzeugen Konflikte
- Risiken und Chancen: die Akzeptanz eines geplanten Systems wird durch unaufgelöste Konflikte gefährdet

3.10.2 Qualitätsaspekte

- Inhalt
- Dokumentation
- Abgestimmtheit

Prüfen des Inhalts

Ziel: Vermeiden, dass die Entwicklung auf falscher Information basiert, d.h. entwickelt wird, was nicht benötigt wird.
Gesucht werden Mängel im Bezug auf

- Vollständigkeit → Alle Bedürfnisse umgesetzt / beschrieben
- Verfolgbarkeit → Identifikation und Quellen
- Korrektheit / Adäquatheit
- Konsistenz → keine Widersprüche
- Keine vorzeitigen Entwurfsentscheidungen
- Überprüfbarkeit → Abnahmekriterium möglich
- Notwendigkeit → Trägt zu einem Ziel bei

Prüfen der Dokumentation

Ziel: Vermeiden, dass Information fehlt oder nicht gemäss geforderten oder vereinbarten Erfordernissen aufbereitet ist
Gesucht werden Mängel im Bezug auf

- Konformität
- Verständlichkeit
- Eindeutigkeit

Prüfen der Abgestimmtheit

Ziel: Vermeiden, dass nicht alle Stakeholder mit den dokumentierten Anforderungen übereinstimmen
Gesucht werden Mängel im Bezug auf

- Abstimmung
- Abstimmung Änderungen
- Konflikte

3.10.3 Prinzipien der Prüfung von Anforderungen

1. Die richtigen Stakeholder beteiligen
2. Fehlersuche und Fehlerkorrektur trennen
3. Aus unterschiedlichen Sichten prüfen
4. Dokumentationsform geeignet wechseln
5. Entwicklungsartefakte konstruieren
6. Prüfung wiederholen

3.10.4 Techniken zur Prüfung von Anforderungen

Für die systematische Prüfung von Anforderungen existieren versch. Techniken, die teilweise auch ergänzend zueinander eingesetzt werden, um Anforderungen möglichst umfassend hinsichtlich festgelegter Prüfkriterien zu überprüfen.

- Stellungsnahme
- Inspektion
- Walkthrough
- Perspektivenbasiertes Lesen
- Prüfung durch Prototypen
- Einsatz von Checklisten

Stellungsnahme

Ziel: dritte Person erstellt eine Expertise bezüglich Qualität

- Der Autor übergibt seine Anforderungen an eine dritte Person
- Identifikation von Qualitätsmängel /Mehrdeutigkeit und Fehler

Inspektion

→ Ist ein sehr formaler Prozess

- Inspektionen haben das Ziel, Entwicklungsartefakte systematisch nach Fehlern zu durchsuchen
- Eine Inspektion besteht aus den Phasen
 - Planung
 - Übersicht
 - Fehlersuche
 - Fehlersammlung
 - Nachkontrolle
 - Reflexion

Dabei kommen unterschiedliche Rollen zum Zug:

- Moderator
- Organisator
- Autor

- Vorleser
 - Inspektoren
 - Protokollführer
- ⇒ als Output resultiert ein **Review-Bericht**

Walkthrough

→ leichtgewichtiges Review / Inspektion

- Weniger Strikt
- Drei Rollen
 - Reviewer
 - Autor
 - Protokollant

Perspektivenbasiertes Lesen

→ Lesen aus unterschiedlichen Perspektiven

- Perspektiven Kunden/Nutzer
- Perspektive Softwarearchitektur
- Perspektive Tester

oder alternativ aus den Perspektiven

- Perspektive Inhalt
- Perspektive Dokumentation
- Perspektive Abgestimmtheit

Prüfung durch Prototypen

- Die Prüfung von Anforderungen durch Prototypen verfolgt den Ansatz, die Anforderungen für den Prüfer erlebbar und damit ausprobierbar zu machen
 - Wegwerfprototypen → wird anschliessend weggeworfen
 - Evolutionäre Prototypen → dieser Prototypen wächst weiter
- Vor der Prüfung muss festgelegt werden, welche Anforderungen mit dem Prototypen überprüft werden sollen
- Vorbereitung
 - Handbuch / Schulung
 - Prüfszenarien
 - Checklisten mit Prüfkriterien
- Prüfung der Anforderungen durch Prototypen kann wie folgt dokumentiert werden
 - Protokoll des Prüfers
 - Beobachtungsprotokoll (durch eine zweite Person)

Einsatz von Checklisten

- tbd ...

3.10.5 Abstimmung von Anforderungen

Die Abstimmung von Anforderungen zielt darauf ab, ein gemeinsames Verständnis der Anforderungen andas zu entwickelnde System unter allen relevanten Stakeholder herzustellen.

Dabei gibt es folgende Aufgaben

- Konfliktidentifikation
- Konfliktanalyse
- Konfliktauflösung
- Dokumentation von Konfliktlösungen

Konfliktanalyse

→ ein identifizierter Konflikt wird auf seine Ursache untersucht

Dabei gibt es unterschiedliche Konflikttypen:

- Sachkonflikt → Mangel an Informationen oder Fehlinformationen
- Interessenskonflikt → geringe Kosten vs. hohe Qualität
- Wertekonflikt → kulturelle Unterschiede
- Beziehungskonflikt → Emotionen, schlechte Kommunikation
- Strukturkonflikt → ungleiche Macht und Autoritätsverhältnisse

Konfliktauflösung

→ alle Stakeholder müssen miteinbezogen werden.

Dabei gibt es folgende Konfliktlösungstechniken:

- Einigung → durch Austausch von Informationen
- Kompromisse → Kombination aus Lösungsalternativen
- Abstimmung → es wird über Alternativen abgestimmt
- Variantebildung → durch Variantenauswahl oder Parametrierung versch. Systemvarianten
- Ober-sticht-Unter → ranghöhrere Partei entscheidet
- Consider all Facts → untersuchen aller Einflussfaktoren, Priorisierung, Relevanz
- Plus Minus Interesting → Positive Negative Auswirkungen der Lösungsvarianten
- Entscheidungsmatrix → Nutzwerkanalyse, evtl. mit Gewichtung

Wenn sich die Lösungsvarianten nur sehr marginal in der Punktzahl unterscheiden, müssen unbedingt eine Sensibilitätsanalyse durchgeführt werden und evtl. weitere Techniken zur Konfliktauflösung eingesetzt werden

| | Lösungs-alternative 1 | Lösungs-alternative 2 | Lösungs-alternative 3 |
|-------------|-----------------------|-----------------------|-----------------------|
| Kriterium 1 | 3 | 6 | 2 |
| Kriterium 2 | 5 | 4 | 10 |
| Kriterium 3 | 10 | 3 | 5 |
| Summe | 18 | 13 | 17 |

Abbildung 3.31: Beispiel einer Entscheidungsmatrix

3.11 Anforderungen verwalten

Anforderungen verwalten gehört zu einer der vier Haupttätigkeiten im RE.

3.11.1 Attribute von Anforderungen

Grundsatz: Um die Anforderungen an ein System über den gesamten Lebenszyklus des Systems hinweg verwalten zu können, ist es notwendig die Informationen zur Anforderung als Attribute möglichst strukturiert zu erfassen.

Attributierungsschema

Die Definition der Attributstruktur für Anforderungen erfolgt über ein Attributierungsschema, das entweder tabellarisch oder in Form eines Informationsmodells definiert werden kann

einige typische Attributtypen werden nachfolgend aufgelistet:

| Attributtyp | Bedeutung |
|---------------|--|
| Identifikator | Kurze, eindeutige Identifikation eines Anforderungsartefakts in der Menge der betrachteten Anforderungen. |
| Name | Eindeutiger, charakterisierender Name. |
| Beschreibung | Beschreibt in komprimierter Form den Inhalt der Anforderung. |
| Version | Aktueller Versionstand der Anforderung. |
| Autor | Benennt den/die Autor(in) der Anforderung. |
| Quelle | Benennt die Quelle bzw. Quellen der Anforderung. |
| Begründung | Beschreibt, weshalb diese Anforderung für das geplante System von Bedeutung ist. |
| Stabilität | Benennt die voraussichtliche Stabilität der Anforderung. Stabilität ist dabei der Umfang, in dem künftig noch Veränderungen bzgl. dieser Anforderung erwartet werden. Mögliche Unterscheidung: »fest«, »gefestigte«, »volatile«. |
| Kritikalität | Im Sinne einer Abschätzung der Schadenshöhe und Eintrittswahrscheinlichkeit. |
| Priorität | Benennt die Priorität der Anforderung hinsichtlich der gewählten Merkmale zur Priorisierung, z.B. »Bedeutung für die Akzeptanz am Markt«, »Reihenfolge der Umsetzung«, »Schaden bzw. Opportunitätskosten durch Nichtrealisierung«. |

Abbildung 3.32: Beispiel gewisser Attributtypen I

| Attributtyp | Bedeutung |
|------------------------------|--|
| Verantwortliche(n) | Benennt die Person, Stakeholdergruppe bzw. Organisation(seinheit), die für diese Anforderung inhaltlich verantwortlich ist. |
| Anforderungstyp | Benennt abhängig vom eingesetzten Differenzierungsschema den Typ der Anforderung (z.B. »funktionale Anforderung«, »Qualitätsanforderung« oder »Randbedingungen«). |
| Status bzgl. des Inhalts | Benennt den aktuellen Status des Inhalts der Anforderung, z.B. »idee«, »Konzept«, »detaillierter Inhalt«. |
| Status bzgl. der Überprüfung | Benennt den aktuellen Status der Validierung, z.B. »ungeprüft«, »in Prüfung«, »überprüft«, »fehlerhaft«, »in Korrektur«. |
| Status bzgl. der Einigung | Benennt den aktuellen Status der Abstimmung, z.B. »nicht abgestimmt«, »abgestimmt«, »konfliktär«. |
| Aufwand | Prognostizierter / tatsächlicher Umsetzungsaufwand dieser Anforderung. |
| Release | Nummer des Releases, in dem die Anforderung umgesetzt werden soll. |
| Juristische Verbindlichkeit | Gibt den Grad der juristischen Verbindlichkeit der Anforderung an. |
| Querbezüge | Benennt die Beziehungen zu anderen Anforderungen. Zum Beispiel wenn bekannt ist, dass die Realisierung dieser Anforderung die vorherige Realisierung einer anderen Anforderung voraussetzt. |
| Allgemeine Informationen | In diesem Attribut können beliebige, für relevant erachtete Informationen zu dieser Anforderung dokumentiert werden. Zum Beispiel wenn die Abstimmung dieser Anforderung auf dem nächsten Treffen mit dem Auftraggeber vorgesehen ist. |

Abbildung 3.33: Beispiel gewisser Attributtypen II

⇒ Attributierungsschema werden dabei häufig projektspezifisch auf Basis bestimmter Rahmenbedingungen definiert bzw. angepasst.

Hierzu gehören:

- Spezifische Merkmale
- Vorgaben
- Vorschriften
- Randbedingungen

3.11.2 Sichten auf Anforderungen

Sichten sind ein reduzierter Zugriff auf die relevanten Anforderungen bspw. nur für bestimmte Rollen oder Personen relevant.

Dabei unterscheiden wir zwischen:

- Selektive Sichten → Darstellung einer Teilmenge der Attributwerte von über definierte Selektionskriterien ausgewählten Anforderungen ⇒ für spezifische Rolle
- Verdichtende Sichten → Darstellung verdichteteter Informationen zu den über definierte Selektionskriterien ausgewählten Anforderungen ⇒ Auswertung

3.11.3 Priorisierung von Anforderungen

Anforderungen werden zu versch. Zeitpunkten in versch. Aktivitäten nach unterschiedlichen Kriterien priorisiert

Vorgehen

Die Vorbereitung der Priorisierung von Anforderungen basiert auf einer einfachen Systematik:

- Bestimmung der Ziele und Randbedingungen der Priorisierung
- Bestimmung der Priorisierungskriterien (Kosten, Risiko, Schaden bei nicht Erfolg, Volatilität, Wichtigkeit)
- Bestimmung der relevanten Stakeholder
- Auswahl der zu priorisierenden Artefakte

Techniken

Es gibt verschiedene Techniken zur Priorisierung.

1. **Ranking** → Rangfolge von Anforderungen wird von ausgewählten Stakeholder ausgewählt
in der Praxis haben relativ schnell, alle Anforderungen Prio 1 erhalten
Man kann den Stakeholder dazu 'zwingen' die Anforderungen in ein Ranking zu verpacken
2. **Top-Ten Technik** → Für ein betrachtetes Kriterium werden die n wichtigsten Anforderungen ausgewählt
3. **Ein-Kriterium-Klassifikation** → basiert auf der Wichtigkeit der Realisierung
nach IEEE-830-1998 drei Prioritätsklassen
 - Mandatory
 - Optional
 - nice-to-have
4. **Kano-Klassifizierung** → Analog dem Kano-Modell
Klassifizierung in
 - Basismerkmale
 - Leistungsmerkmale
 - Begeisterungsmerkmale
5. Wiegers'sche Priorisierungsmatrix → sehr aufwändig, wird nicht weiter betrachtet

3.11.4 Verfolgbarkeit von Anforderungen

Im Rahmen der Verwaltung von Anforderungen werden Verfolgbarkeitsinformationen von Anforderungen aufgezeichnet, organisiert und gepflegt und dies über den gesamten Lifecycle

Definition: Information auf der Basis eines klar definierten Verwendungszweckes und wird später in der Systementwicklung oder Systemevolution gebraucht

Nutzen

- Vereinfachung der Nachweisbarkeit
- Identifikation von unnötigen Eigenschaften im System
- Identifikation von unnötigen Anforderungen
- Unterstützung der Auswirkungsanalyse (→ Entwicklungsartefakte)
- Unterstützung der Wiederverwendung
- Unterstützung der Festlegung der Zurechenbarkeit (Zuordnung des Aufwandes, Nachkalkulation)
- Unterstützung der Wartung und Pflege

Klassifikation von Verfolgbarkeitsbeziehungen

hinsichtlich der Verfolgbarkeitsbeziehungen von Anforderungen werden drei Klassen von Verfolgbarkeitsbeziehungen unterschieden:

- Pre-Requirements-Specification-Traceability
- Post-Requirements-Specification-Traceability
- Traceability zwischen Anforderungen

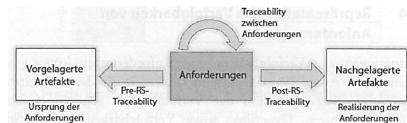


Abbildung 3.34: Abbildung der Klassen der Traceability

3.11.5 Versionierung von Anforderungen

Die Versionierung und Konfiguration von Anforderungen ermöglicht es, über den Lebenszyklus eines Systems oder Produktes hinweg, spezifische Entwicklungsstände (von Anforderungen und Anforderungsdokumenten) verfügbar zu halten.

Konfiguration von Anforderungen

Eine Anforderungskonfiguration fasst eine definierte Menge logisch zusammengehöriger Anforderungen zusammen, wobei jede Anforderung maximal in einer Version in der Anforderungskonfiguration enthalten ist. Dabei unterscheidet man zwischen zwei Dimensionen:

- Produktdimension: die einzelnen Anforderungen der Anforderungsbasis
- Versionsdimension: die versch. Versionsstände einer Anforderungen

Anforderungsbasislinien

Anforderungsbasislinien sind ausgezeichnete Anforderungskonfigurationen, die stabile Versionen von Anforderungen umfassen und oftmals auch Auslieferungsstufen des Systems definieren

- Grundlage zur Planung von Auslieferungsstufen
- Abschätzung des Realisierungsaufwandes
- Vergleichen mit Konkurrenzprodukten

3.11.6 Verwaltung von Anforderungsänderungen

Über den gesamten Lebenszyklus eines Systems hinweg verändern sich die Anforderungen. Die Änderungen an den Anforderungen werden in einem systematischen Änderungsmanagementprozess verwaltet und bearbeitet.

Change Control Board

Im Änderungsmanagementprozess ist das Change-Control-Board (CCB) für die Bearbeitung eingehender Änderungsanträge verantwortlich.

Die Aufgaben des CCB sind:

- Klassifikation eingehender Änderungsanträge
- Bestimmung des Aufwands einer Änderung
- Beurteilung der Änderungsanträge hinsichtlich Aufwand / Nutzen
- Definition neuer Anforderungen auf Basis eingehender Änderungsanträge
- Entscheidung über Annahme oder Ablehnung eines Änderungsantrags
- Priorisierung der angenommene Änderungsanträge
- Zuordnung der Änderungen zu Änderungsprojekten

Typische Vertreter sind:

- Änderungsmanager
- Auftraggeber
- Architekt
- Nutzervertreter
- Qualitätsbeauftragter
- Anforderungsingenieur

Änderungsantrag

Für notwendig erachtet Änderungen von Anforderungen werden in Form von Änderungsanträgen dokumentiert und an das CCB übermittelt.

Dabei umfasst der Antrag mind. folgende Informationen:

- Identifikator des Änderungsantrags
- Titel
- Beschreibung der notwendigen Änderung
- Begründung
- Datum
- Antragssteller
- Priorität

- Zusätzliche Informationen
 - Prüfer der Änderung
 - Status der Auswirkungsanalyse
 - Status der Entscheidung CCB
 - Priorität CCB
 - Verantwortlicher für Umsetzung
 - Systemrelease

Klassifikation

Drei Arten eines Änderungsantrags

- Korrektive Änderungen → Fehlverhalten
- Adaptive Änderungen → Anpassung des Systems
- Ausnahmeänderungen → Hotfix, muss unmittelbar umgesetzt werden

Dabei sieht das Vorgehen für korrektive und adaptive Änderungen wie folgt aus:

- Auswirkungsanalyse und Beurteilung der Änderung
- Priorisierung der Anforderungsänderung
- Zuordnung der Änderung zu einem Änderungsprojekt
- Kommunikation der Annahme / Ablehnung des Änderungsantrages

3.12 Werkzeugunterstützung

3.12.1 Allgemeine Werkzeugunterstützung

Viele Systementwicklungswerkzeuge können auch RE unterstützen

- Testverwaltungswerkzeuge
- Konfigurationswerkzeuge
- Wiki
- Bürossoftware Visualisierungswerkzeuge

3.12.2 Modellierungswerkzeuge

Tool-Suiten

- UML / BPMN / TOGAF: Enterprise Architect (Sparx)
- IBM Rational Tool Suite
- Magic draw
- ...

Features

- UML Modellierung
- Model Driven Architecture (MDA)
- Reverse Engineering
- Standard Schnittstellen
- ...

3.12.3 Requirements-Management-Werkzeuge

Ein Requirements-Management-Werkzeug sollte dabei folgende grundlegende Eigenschaften aufweisen

- Verschiedene Informationen verwalten
- logische Beziehungen zwischen Informationen verwalten
- Jedes Artefakt eindeutig identifizieren
- Informationen flexibel und sicher zugänglich machen, bspw. durch Zugriffskontrolle
- Sichten auf die Informationen unterstützen
- Informationen organisieren bspw. durch Attributierung und Hierarchiebildung
- Berichte über die Informationen erstellen
- Dokumente aus den Informationen generieren

Spezialisierte Requirements-Management-Werkzeuge (bspw. IBM Rational RequisitePro bzw. IBM Rational DOORS)

Charakteristische Eigenschaften dieser Werkzeuge sind:

- Verwaltung von Anforderungen und Attributen auf der Basis von Informationsmodellen
- Organisation von Anforderungen (mittels Hierarchieebenen)
- Konfigurations- und Versionsmanagement auf Anforderungsebene
- Definition von Anforderungsbasislinien
- Mehrbenutzerzugriff und -verwaltung (bspw. Zugriffskontrolle)
- Verfolgbarkeitsmgmt (Traceability Management)
- Konsolidierung der erfassten Anforderungen (bspw. Sichtenbildung)
- Unterstützung des Änderungsmanagement (Änderungskontrolle)

3.12.4 Werkzeugeinführung

Erst nach der Einführung von RE-Vorgehensweise und Techniken kann ein passendes Werkzeug ausgesucht werden. Diese Einführung setzt klare Verantwortlichkeiten und Vorgehensweise im RE voraus. Dabei sind folgende Gesichtspunkte zu beachten:

- Benötigte Ressourcen planen
- Risiken durch Pilotprojekte umgehen
- Evaluierung anhand von definierten Kriterien
- Über Lizenzkosten hinausgehende Kosten berücksichtigen
- Benutzer schulen

3.12.5 Beurteilung von Werkzeugen

Die Bewertung lässt sich auf sieben Sichten strukturieren

- Projektsicht → Unterstützung der Projektplanung
- Benutzersicht → insbesondere Bedienung
- Produktsicht → Funktionalität
- Prozesssicht → methodische Unterstützung
- Anbietersicht → Service des Anbieters
- Technische Sicht → Interoperabilität, Skalierbarkeit
- betriebswirtschaftliche Sicht → Kosten

⇒ Für jede Sicht sind klare Kriterien zu definieren

Kapitel 4

Software Architektur

Es gibt durchaus parallelen zur klassischen Architektur → gute Architektur:

- utilitas (Nützlichkeit)

Die Software erfüllt die **funktionalen und nicht-funktionalen Anforderungen** der Nutzer und Kunden

- firmitas (Festigkeit)

Die Software ist stabil im Hinblick auf die **geforderten Qualitätseigenschaften** z.B. langlebig, da zukünftige Weiterentwicklungen möglich sind, ohne das System komplett neu zu bauen

- venustas (Schönheit)

Die Software ist sowohl **aussen** (gegenüber dem Nutzer) wohl strukturiert, sodass sie intuitiv nutzbar ist, als auch **innen** (gegenüber demjenigen, der die Software pflegen und weiterentwickeln soll) wohl strukturiert, sodass dieser die internen Strukturen der Software leicht verstehen und damit gut seinen Aufgaben nachkommen kann

4.1 Grundlagen

Der CHOAS-Report der Standisch Group zeigt, dass wir es immer noch nicht schaffen, wiederholbar qualitativ **hochwertige Software** zu erschwinglichen **Kosten** und im vorgegebenen Zeitfenster mit der notwendigen **Funktionalität**

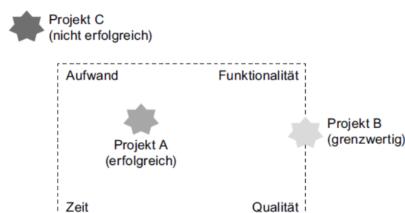


Abbildung 4.1: Abbildung des magischen Dreiecks zwischen Aufwand, Zeit, Funktionalen, Qualität et

4.1.1 Definitionen

Was ist Softwarearchitektur?

- RE und Architekturentwurf zwei zentrale Schlüsselfaktoren
- Bei beiden ist das Risiko von gravierenden Fehlentwicklungen hoch

System: A collection of components organized to accomplish a specific function or set of functions
Software: Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system
Softwareintensives System

- Menge von Bausteinen
- Bausteine, die aus Software bestehen übernehmen wesentliche Aufgaben zur Erfüllung des Systemszweck

Softwarearchitektur: Die Softwarearchitektur definiert die grundlegenden Prinzipien und Regeln für die Organisation eines Systems sowie dessen Strukturierung in Bausteinen und Schnittstellen und deren Beziehungen zueinander wie auch zur Umgebung. Dadurch legt sie Richtlinien für den gesamten Systemlebenszyklus, angefangen bei Analyse über Entwurf und Implementierung bis zu Betrieb und Weiterentwicklung, wie auch für die Entwicklungs- und Betriebsorganisation fest.

Schnittstelle:

- repräsentiert einen wohldefinierten Zugangspunkt zum System oder dessen Bausteinen
- Beschreibt Eigenschaften dieses Zugangspunkts (bspw. Attribute, Daten und Funktionen)

4.1.2 Grundlegende Konzepte

- Es ist eine Art Bauplan für die Festlegung der Struktur und Konzepten
- Hierarchische Zerlegung in eine Menge von Bestandteilen

4.1.3 Bausteine

Beinhaltet sämtliche Software- oder Implementierungsartefakte, die letztendlich Abstraktionen von Quellcode darstellen

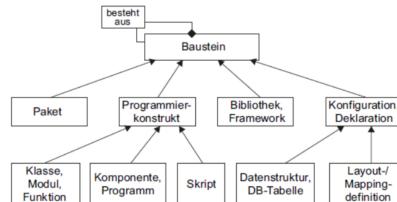


Abbildung 4.2: Abbildung Bausteine

- Bausteine bieten Schnittstelle an, welche er im Sinne eines Vertrags garantiert
- Über die angebotene und benötigten Schnittstellen kapselt der Baustein die Implementierung dieser Schnittstelle
 → Dadurch kann er durch einen anderen Baustein ersetzt werden, die dieselben Schnittstellen exportieren und gegebenenfalls importieren
- Bausteine können auch hierarchische (De-)Kompositionen beinhalten
- **Blackbox-Sicht** man sieht nur die exportierten und importierten Schnittstellen → respektiert das Geheimnisprinzip
 ⇒ Sicht des **Bausteinnutzers**
- **Greybox-Sicht (auch: Konfiguration)** zeigt wie importierte Schnittstellen mit den exportierten Schnittstellen verschaltet werden
 ⇒ Sicht des **Bausteinkonfigurators**
- **Whitebox-Sicht (auch: Glassbox-Sicht)** erlaubt Blick ins innere
 ⇒ Sicht des **Bausteinimplementierers**

4.1.4 Schnittstellen

- Standardschnittstelle

Wird von ausserhalb definiert. Sowohl der importierende als auch der exportierende Baustein halten sich daran

- Angebotene Schnittstelle

definiert durch den Exporteur, neben Standardschnittstelle die meist verwendete

- Angeforderte Schnittstelle

definiert durch den Importeur, häufig bei Frameworks anzutreffen

- unabhängige Schnittstelle

Importeur und Exporteur hat eigene Schnittstelle, es braucht einen Adapter(-Pattern)

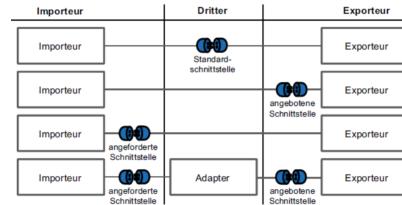


Abbildung 4.3: Abbildung Schnittstellenvarianten

4.1.5 Softwarearchitekturbeschreibung

- Dabei wird das System von seiner Umgebung beeinflusst und umgekehrt
- Jedes System hat eine Architektur
- Die Architektur wird laut Standard durch eine einzige Architekturbeschreibung beschrieben
- Darüber hinaus hat ein System eine Reihe von Interessenvertretern (Stakeholder)
- Die Stakeholder haben eine Menge von Anliegen
- Die Architekturbeschreibung nimmt die Anliegen der Stakeholder auf und begründet damit die getroffenen Architekturentscheidungen in den Begründungen

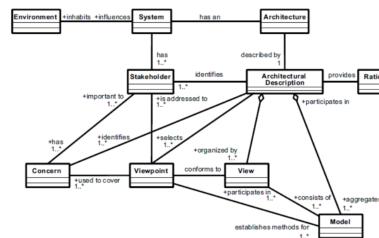


Abbildung 4.4: Abbildung Kernelemente des konzeptionellen Modells gemäss IEEE-Standard 1471-2000

Architektsichten

Architekturbeschreibungen sind Sichten auf die Architekturbeschreibung nach IEEE gibt es Standard-Sichten:

- functional view
- physical view
- technical view

Diese Sichten sind im Verständnis von Philippe Kruchten *Architectural Blueprints*

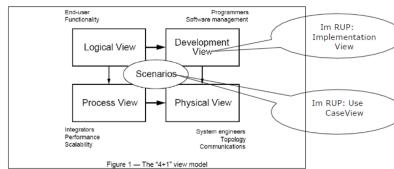


Abbildung 4.5: Abbildung Architectural Blueprints

Architekturebene

Eine Architekturbeschreibung besteht dabei aus einer Menge von Architekturebenen

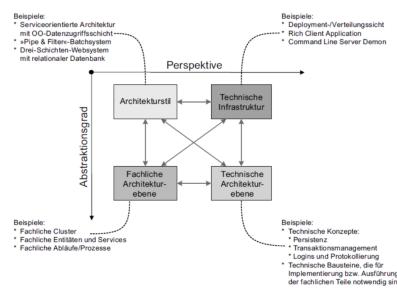


Abbildung 4.6: Abbildung Architekturebene

- Architekturebene können für sich alleine betrachtet werden, allerdings beeinflussen sie sich gegeneinander und sind somit voneinander abhängig
- Der **Architekturstil** ist dabei die zentrale Architekturmetapher des Systems
Drei-Schichten-Architektur unter Verwendung eines Model-View-Controllers in der Präsentationsschicht und einem objektrelationalen Mapping in der Datenhaltungsschicht strukturiert
- **technische Infrastruktur** hingegen fixiert die Netzwerkschnittstelle der Architektur

Wir haben einen Thin Client mit einem Web- und Application Container und einer relationen DB

Wechselwirkung

- Die Architektur wird von der Umgebung beeinflusst und umgekehrt
- Folgende Beispiele der Wechselwirkung werden aufgelistet
 - Projektumfeld und Projektmanagement
 - Produktmanagement und RE
 - Ausführungsplattform und Betrieb
 - Werkzeugumgebung und Entwicklung

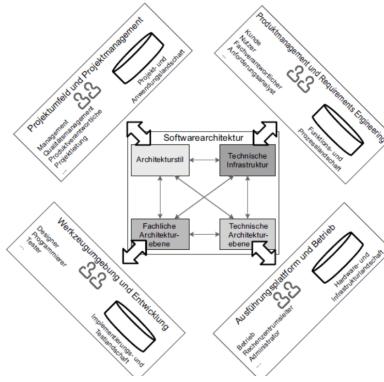


Abbildung 4.7: Abbildung des Gesamtbildes

4.1.6 Qualität und Nutzen der Softwarearchitektur

- Die Qualität der Softwarearchitektur kann nur im Kontext konkreter Qualitätsziele bewertet werden
- Diese Qualitätsziele werden von den Anforderungen, insbesondere von den nichtfunktionalen Anforderungen und geforderten Qualitätseigenschaften abgeleitet und sind somit spezifisch für das konkrete Softwaresystem
- Qualitäten lassen sich nur indirekt durch Merkmale beschreiben

Qualitätsmerkmale

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

Twin Peak Model

Zeigt die Softwarearchitektur aus der Vogelperspektive

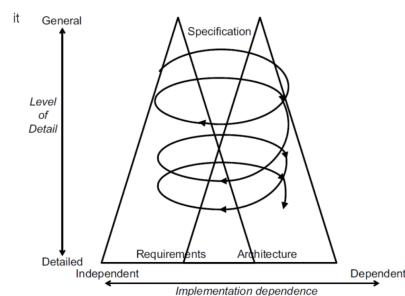


Abbildung 4.8: Abbildung der Architektur aus der Vogelperspektive

4.1.7 Softwarearchitekturentwurf

Architecture Business Cycle

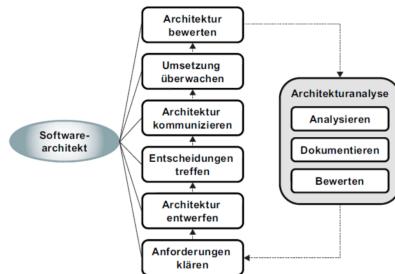


Abbildung 4.9: Ablauf des Architecture Business Cycle

Alternatives Modell

- Anforderungen und Randbedingungen analysieren
funktionale und nichtfunktionale Anforderungen untersuchen
Qualität und Stabilität überprüfen
Lücken in Anforderungen aufdecken
erstes Verständnis für den Architekturstil
- Architektsichten und technische Konzepte entwerfen
Architektur detaillierter ausarbeiten
Sichtenbasierte Beschreibung der unterschiedlichen Architekturebene
funktionale Anforderungen auf fachliche Architekturebene herunterbrechen
- Architektur und Entwurfsentscheidungen bewerten
erarbeitete Architektur muss qualitätsgesichert werden
konkrete Szenarien erarbeiten
- Umsetzung begleiten und überprüfen
Kommunizieren von Entwickler bis zum Kunden

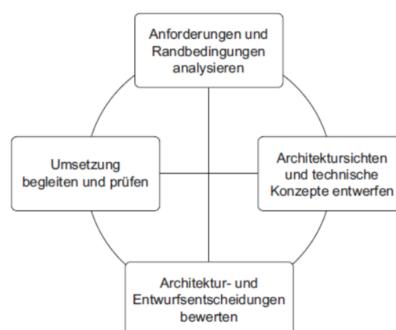


Abbildung 4.10: Ablauf des alternativen Modells

4.1.8 Der Softwarearchitekt

- kommuniziert gerne
- Hat Kontakt zu allen anderen Stakeholder
- ist verantwortlich, dass das System den sicherheitstanforderungen genügeträgt
- Machbarkeit der Anforderungen abklärt
- Priorisiert die funktionalen und nichtfunktionalen Arbeiten
- Zeit Integrationsmöglichkeiten
- erarbeitet, evaluiert und bewertet Lösungsansätze
- Zentraler Ansprechpartner
- Grosse Strukturierung des Systems
- Integriert neue Technologien
- Verantwortlich der Programmierrichtlinien
- unterstützt Tester
- erklärt Architektur und gibt Erfahrung an Entwickler weiter

4.2 Entwurf von Softwarearchitekturen

4.2.1 Entwurfsprinzipien

4.2.2 Architekturzentrierte Entwicklungsansätze

4.2.3 Architekturmuster

4.2.4 Entwurfsmuster