Zusammenfassung Requirements Engineering

Philipp Jäcks

18. Januar 2016

Inhaltsverzeichnis

1 Einleitung

Requirements sind nicht gegeben, sondern müssen erarbeitet werden und im Laufe des Entwicklungsprozess verfeinert und verbessert werden. Interdisziplinäre Disziplin mit Menschenkontakt. Abbildung der informalen Welt der Stakeholder auf die formale Welt der Entwickler/Verhalten der Software.
In RE muss der Analyst verstehen:

- System-as-is
- System-to-be
- System-to-be-next

Daraufhin wird ein Model der aktuellen Situation erstellt und das in ein Model der vorgestellten Situation überführt. (Grob zsmf)

1.1 non-functional R

- Bedingungen die an das System als Ganzes gestellt werden, z.B. Zeitbeschränkungen, Performance, Standards, den Entwicklungsprozess
- **Aufteilung in**
  - *Product* Effizienz (Performance, Speicherbedarf), Usability, Zuverlässigkeit
  - *Organisational* Standards, Entwicklungsprozess
  - *External* Ethische, Gesetzliche, Interoperabilität
- oftmals kritischer als functional, da User bei Fehlverhalten eines Dienstes (=functional R) einen Weg drum herum findet. Bei fehlerhafter NFR ist dies nicht so leicht möglich.
- quantisierbar spezifizieren um objektiv testen zu können. Dazu folgende Kriterien

Eigenschaft	Messeinheit
Speed	Operationen/Sekunde; Reaktionszeit auf Usereingabe
Ease of Use	Einarbeitungszeit; Hilfsfenster
Zuverlässigkeit	Durchschnittszeit bis Fehler eintritt
Robustheit	Was passiert mit Daten nach Fehler? Wie schnell lässt sich das System nach Fehler neus

1.2 functional R

- Aussagen über Dienste die das System liefern muss, speziell konkrete Features, wie reagiert das System auf Eingaben, bei bestimmten Situationen.
- Manchmal Aussage darüber was das System in Situation grade *nicht* machen soll.

- **Aspekte:**
- *Data:* Struktur, Verwaltung, Zugriff, Übertragung
- *Functions:* Input, Output, Verarbeitung
- *Verhalten:* beobachtbares Verhalten des Systems (auch in Fehlersituationen)

Aufteilung der Anforderungsanalyse in RS (vglbar mit Lastenheft) und SRS (vglbar mit Pflichtenheft).

1.3 Domain Requirements

Ergeben sich aus der Anwendungsdomäne des Systems und spiegeln die Charakteristik und Bedingungen der Domäne wieder. Kann sowohl non-functional als auch functional R sein.

1.4 Requirements Specification (RS)

Product Constraints bestimmen

1. Zweck des Produkts
   Aufgabe, Ziel, Was soll das Produkt bringen? Wichtige Punkte
   - Welchen Vorteil liefert das Produkt?
   - Ist der Vorteil messbar?
   - Ist der Aufwand für den Vorteil rechtfertigbar?
2. Client, Kunde, Stakeholder
3. Nutzer des Produkts
4. Requirements Bedingungen
5. Definitionen, Nomenklatur
6. relevante Fakten
7. Annahmen

Functional Req. bestimmen

- Scope des Produkts
  The features and functions that characterize a product, service, or result.
- Functional and Data Req
  Spezi der Dienste und Datenstruktur

Non-Functional Req. bestimmen

1.5 Software Requirements Specification (SRS)

Spezifikation für ein Softwareprodukt.
Eigenschaften einer guten SRS:

- Korrekt, Komplett, Konsistent, Verifizierbar...
- eindeutig -> jede spezifizierte Anforderung hat genau eine Interpretation
- Grundlagen, die eine SRS betrachten/beantworten sollte
  - Functionality: Was soll die Software leisten?
  - External Interfaces: Interaction with people
  - Performance: Geschwindigkeit, Verfügbarkeit, ...
  - Attributes: Portabilität, Korrektheit, Wartbarkeit, Sicherheit...
  - Design Contraints for the impl:

1.6 Main Activities in RE

- Erhebung der R. (informal)
  Welche Probleme bewältigen? Wer ist Kunde, Stakeholder, Nutzer? Techniken: Interviews, Brainstorming, Prototyping
- Model and Analysing der R -> Ergebnis: Formale R
- Kommunikation der R
  Formale R müssen auch für Stakeholder verständlich sein
- Abgleich/Verifikation der formalen R mit Vorstellungen der Stakeholder
- Weiterentwicklung der R
  Vorstellung des Softwaresystems kann Stakeholder/Nutzer/Kunde dazu veranlassen R umzuformulieren

2 Use Case Analysis

Ziel: Abbildung der Realität in Anwendungsfalldiagramm. Dabei wird in der Regel ein konkretes Szenario beschrieben.

- Use Case = Menge von Aktionssequenzen, die ein Nutzer oder externes System ausführt um ein bestimmtes Ziel zu erreichen. Beschreibt inhaltlich was beim Versuch der Zielerreichung passieren und schiefgehen kann.
- Helfen in der **Planungsphase**:
  - verdeutlichen wichtige Ziele des Produkts
- in der **Entwicklungsphase**
  - fassen Kernpunkte eines Features für Entwickler gut zusammen
- Erfassen der funktionalen R aus Sicht des Nutzers

Actor:

- interagiert mit dem System; entweder Person oder wieder ein System
- gibt Input und erhält Output vom System
- extern; keine Kontrolle über den Use Case
- Vererbung möglich
- **Primär** und **Sekundär** Actor
  - Primär: will ein Ziel mithilfe des Systems erreichen
  - Sekundär: wird vom System benötigt um das Ziel des Primären zu erreichen

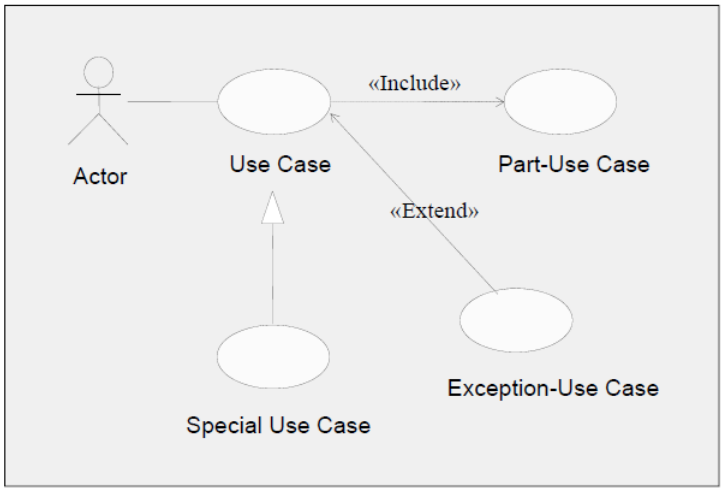


Abbildung 1: Elements of Use Case

Use Case begleitet die ganze Entwicklung. Wird *realisiert* vom Design Model, *Implementiert* vom Implementation Model und *Verifiziert* vom Test Model.

Flow of Events

- **Happy Path** normal basic Flow
- alternative flows
  - exception flows for error situations
  - reguläre Alternativen ein Ziel zu erreichen
  - odd cases

Scenario = Sequenz von Aktionen die die Interaktion zwischen Actor und System beschreibt (quasi ein Element des Use Case)

Use Case Reports
Beschreibt den Use Case im Detail. Einige Bestandteile nach Cockburn

- Ziel
- Context of Use: In welchem technischen Kontext tritt der Use Case auf?
- Level: Level of Detailt, Summary, Primary Task etc
- Folgende sind optional
- Actor
- Stakeholder
- Conditions
- Trigger

Report ist wichtiger als Use Case Model (<= Das Diagramm).
Quasi Semi-Structured textuelle Beschreibung

- Use Case Name
- Primary Actor
- Stakeholder and Interests
- Preconditions
- Success Garantie
- Trigger
- Main Success Scenario
- Extensions (zum main scenario)

Dazu gehören noch zusätzliche Spezi (mit Usability, Zuverlässigkeit Performance etc pp, quasi NFR zum Use Case) und Glossar -> Begriffsklärung.

Use Case Analysis

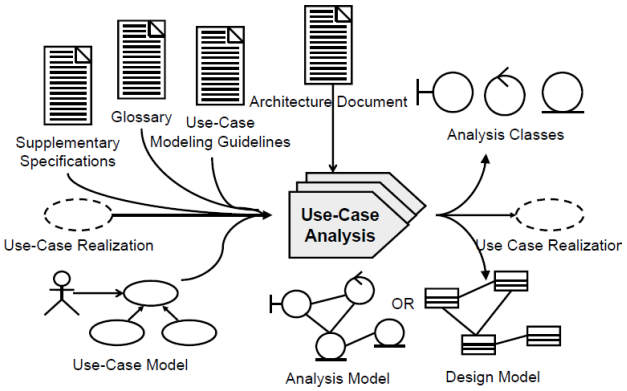


Abbildung 2: Use Case Analysis

Summary

Vorteile	Nachteile
zeigen functional R in einfacher, verständlicher Art Erstellen Framework[1] für non-functional R und Projekt Details.	zeigen ausschließlich functional R

2.1 Glossar

- **Sequenzdiagramm** = Graphische Darstellung eines Szenarios, das Interaktion zw. Objekten zeitlich darstellt

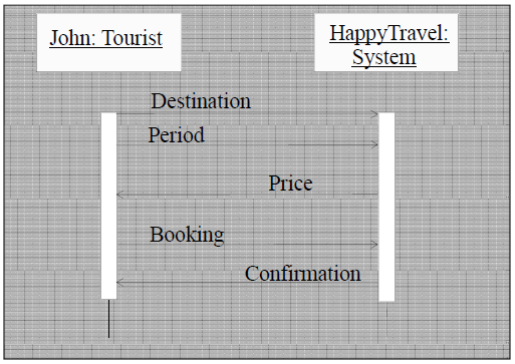


Abbildung 3: Ex Sequence Diagram

- **Collaboration Diagram** = Interaktionsdiagramm - zeigt eine Sequenz von Nachrichten, die eine Operation oder Transaktion implementieren

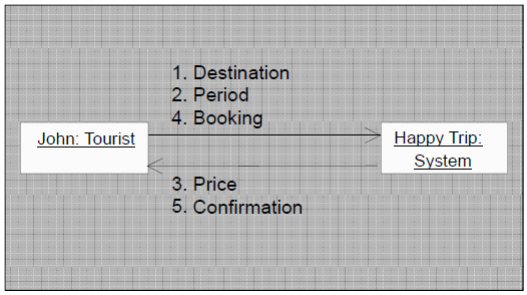


Abbildung 4: Ex Collaboration Diagram

3 Requirements Elicitation - "Anforderungsanalyse"

- Prozess des *Suchens, Aufdeckens, Erwerbens, Ausarbeiten* von R für computerbasierte Systeme
- Teil der frühen Entwicklungsphase, aber fortlaufend und kritisch
- komplex, verschiedenste Techniken - vor allem aus Sozialwissenschaften - finden Anwendung
  - großer Problemfaktor liegt in Kommunikation zw. Engineers und Kunde
  - Iterative Erarbeitung, setzt stark auf Kommunikationskills der Engineers und Bereitschaft der Stakeholder
  - Problem: Konzepte, die für A verständlich sind, könnten für B komplett unverständlich sein
- Probleme der Erhebung
  - Scope: Systemgrenzen sind falsch definiert; unnötige technische Details spezifiziert, die eher verwirren als helfen
  - Understanding: Kunde weiß nicht genau was er will; was seine Computer-Umgebung her gibt; versteht das Problem nicht ganz; Lässt Infos aus die "offensichtlich" sind0
  - Flüchtigkeit: R verändern sich mit der Zeit

5 Fundamentale Aktivitäten zur Erhebung

1. Verstehen der Anwendungsdomäne
   Beschreibung existierender Arbeitsprozesse und die zugehörigen Probleme, die vom System gelöst werden sollen.
2. Quellen der R identifizieren
   verschiedenste Quellen -> Stakeholder (meistens); Existierende Systeme und Prozesse; exist. Doku über aktuelles System und Prozesse
3. Analyse Stakeholder
   Identifizieren wer der "offensichtlichste" Stakeholder ist -> mal der Endkunde, mal der Geldgeber etc
4. Methoden, Ansätze, Tools auswählen
   Gründe für Methodenauswahl: einzige, die der Analyst kennt; sein Favorit; vorgeschrieben;...
5. R erheben von Stakeholder und anderen Quellen
   Ergebnis = detaillierte Menge von R in natürlicher Sprache und einfachen Diagrammen

Wir benötigen unterschiedliche Techniken, da jede Technik anderes Wissen als Ergebnis liefert.

	Interviews	Domain	Groupwork	Ethnography	Prototyping	Goals	Scenarios	Viewpoints
Understanding the domain	x	x	x	x		x	x	x
Identifying sources of requirements	x	x	x			x	x	x
Analyzing the Stakeholders	x	x	x	x	x	x	x	x
Selecting techniques and approaches	x	x	x					
Eliciting the Requirements	x	x	x	x	x	x	x	x

Abbildung 5: Which techniques and approaches should be used for a given requirements elicitation activity?

3.1 Techniken und Ansätze

3.1.1 Task Analysis and Domain Analysis

Beschreibung der Aufgaben, Rollen und Aufgabendomain in der aktuell vorgestellten Situation.

Task Modeling

- *Hierarchisches Beschreibung* in SubTasks und deren Beschreibung
- Tasks werden ausgeführt um ein Ziel zu erreichen; meist von Actor in bestimmter Rolle
- Alternative: *Temporale Beschreibung*

Task Domain
oft objektorientiert modelliert

Wurzel eines Task Models ist ein Use Case

Analysis Synthesis Bridge Model

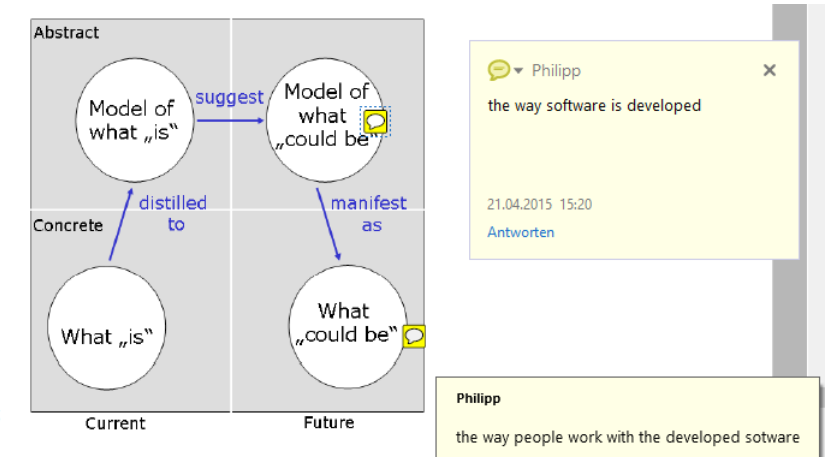


Abbildung 6: Analysis Synthesis Bridge Model

3.1.2 Repertory Grids

- liefert impliziertes Wissen
- basiert auf den persönlichen Konstrukten eines Individuums, in denen die Interaktion mit anderen eingeordnet wird -> hat Einfluss auf die weitere Interaktion mit anderen Menschen
- Zuordnung von Werten zu einer Domain Entität
  Ergebnis => System wird als Matrix modelliert bei der die Elemente des Systems kategorisiert werden
- Ziel ist das Identifizieren von Unterschieden und Gemeinsamkeiten zwischen den einzelnen Domainbereichen

3.1.3 Interviews

- meist verbreitet und oft eingesetzt
- Human Based social Activity -> eher informal und Ergebnis hängt stark vom Zsmspiel der Beteiligten ab
- große Datenmenge in kurzer Zeit mgl
- 3 Arten:
  - unstructured: kein vordefinierter Ablauf von Fragen etc; oft eingesetzt wenn Domäne etc noch sehr unklar formuliert; Risiko das manche Themen komplett wegfallen

- semi-structured: Zwischending
- structured: vorgefertigter Fragenkatalog, der abgearbeitet wird; Erfolg hängt von den richtigen Fragen ab, wann und wem sie gestellt werden

3.2 Contextual Design Approach

Grundidee: Daten vom Kunden sammeln und daraus entscheiden was das System können soll
Ablauf in 2 Phase

1. Kontextuelle Anfrage:
   - Kombination aus verschiedenen Elicitation Techniken (Interview, Observation)
   - Führende/Lenkende Prinzipien
     - Kontext: analysieren des Kundenarbeitsplatzes
     - Partnership: Analyst sucht nach Strukturen/Abläufen in der Arbeit; Kunde erläutert wie Arbeit tatsächlich abläuft (Technik: Kunde bringt dem Analysten den Arbeitslauf bei)
     - Interpretation: Analyst interpretiert gesammelte Daten über Arbeitsplatz und gleicht mit dem Eindruck des Kunden ab
     - Fokus: jede Elicitation Technik fokusiert auf einen Teilaspekt. Am Ende muss das "Große Ganze" daraus entstehen
2. Erstellung von 5 Work Models
   - Flow Models: Beschreiben Arbeitsteilung und -koordination aus Sicht eines Arbeiters
     oft: mehrere Flow Models (aus unterschiedlichen Perspektiven) benötigt
   - Sequence Models: Beschreibt konkrete Aufgaben der Arbeit
     Zweck, Trigger, Arbeitsschritte, Unterbrechungen, Probleme,
   - Artefact Models: beschreibt Artefakte
     Wann/von wem erstellt?; Struktur; Inhalt des Artefakts; Wie werden sie präsentiert?
   - Cultural Models: Organisationskultur -> beschreibt Erwartungen, Wünsche der Arbeiter; organisatorische Vorschriften
   - Physical Models: beschreibt den physikalischen Arbeitsplatz -> Aufbau; Bewegung innerhalb; eingesetzte Technik;
3. Analyse der 5 Modelle um die 'System-to-be' zu bestimmen; Quasi das Redesign der Arbeit erbringt das Design der Software

4 Erweitern der Requirements Erkenntnisse

RE = koordinierte Menge von Aktionen zum Entdecken, Evaluieren, Dokumentieren, Konsolidieren, Überarbeiten von Zielen, Bedingungen, Annahmen, Fähigkeiten die das 'System-to-be' erfüllen sollte um Probleme des 'System-as-is' zu lösen und neue Möglichkeiten zu liefern.

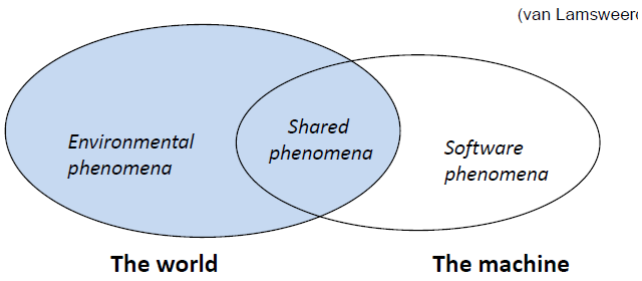


Abbildung 7: Why, What and Who Dimension of RE

R beziehen sich auf die Umwelt und betreffen die Maschine nur indirekt.

- *System R*: vorgeschrieben Angaben, die das 'System-to-be' u.U. mit anderen System Komponenten, leisten kann
- *Software R*: vorgeschriebene Angaben, die **ausschließlich** das 'System-to-be' leisten kann;
  - Domain Properties: Invariante Bedingungen
  - Annahmen: müssen erfüllt werden von Umwelt

4.1 Why, What and Who Dimension of RE

- **Why-Dimension** kontextbezogene Gründe für ein neues System müssen gegeben werden - in Bezug auf die Ziele, die das neue System erfüllen soll
  Ziele anhand der Beschränkungen des aktuellen Systems herausarbeiten
  Zielfindung liefert oft Konflikte, da Ziele aus unterschiedlichen Perspektiven def. werden
- **What-Dimension** Bestimmen der Dienste, Bedingungen, Annahmen des System-to-be anhand der bestimmten Ziele mit Hilfe von Elicitation Techniken
- **Who-Dimension** Bestimmt die Kompetenzen, die notwendig sind um Ziele zu erreichen

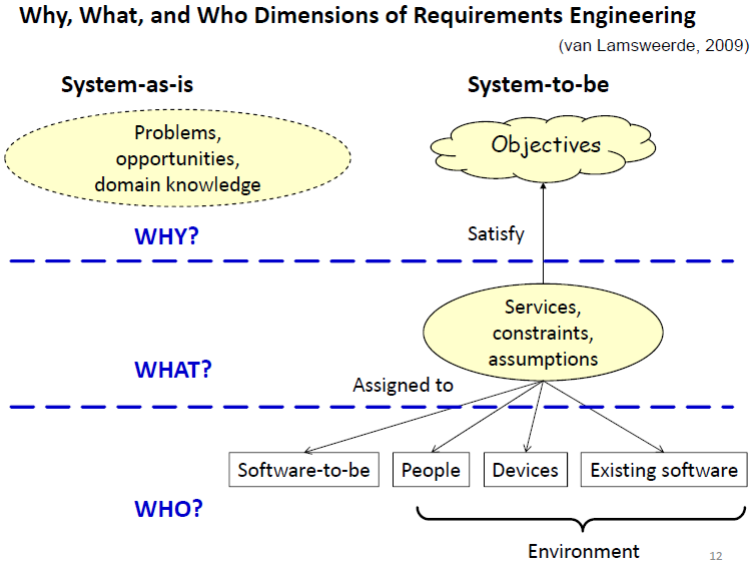


Abbildung 8: Why, What and Who Dimension of RE

4.2 Darstellung der R

- natürliche Sprache für einzelne R nutzen
  Verbesserung durch: Strukturierung der R; Regeln für das beschreiben (z.b. Verbot bestimmter Wörter); Redundanzprüfung
  Limitierung: nicht zwingend eindeutig; Verknüpfung zw R kann schwer zu ziehen sein
- Graphische Notation, die Daten, Funktionen, Verhalten beschreiben
- Formale Modelle
- Formalisierung R durch
  Designation = Formale Grundterm (e.g. Prädikat) durch informelle Beschreibung erklären
  Definition Anhand definierter Termine neue Termine def.; lässt sich besser begründen, aber ist eingeschränkter in der Beschreibung

4.3 Zusammenfassung

- R beschreibt Bedingung über ein Phänomen der Umwelt
- Beschreibung der R muss die Trennung zw. Maschine und Umwelt respektieren und zw
  Umwelteigenschaften, die gegeben sind (indicative) und die von der Maschine geleistet werden (optative)

5 Z-Notation

- basiert auf Mengenlehre und Prädikatenlogik 1. Stufe
- beschreibt Zustände (durch Menge, Objekte, Prädikate) und Übergänge (durch Beschreibung des Zustands vor und nach Übergang) des Systems
- Schema beschreibt Zustand bzw. Übergang
- strukturiertes Wissen über Domäne wird als Ausgangspunkt zum Modellieren benutzt - *Domäne* = exist. und aufgestellte Systeme die ähnliche Ziele haben
  Aufgestelltes System = System (Entitäten die miteinander interagieren um ein Domainziel zu erreichen) + Umwelt(Elemente, die Input geben und auf Output reagieren)
- **Modelling the domain**
  - als Graph; V = Aussagen über zu erreichende Ziele; formale Spezi, das Verhalten des Systems beschreibt; Erklärungen (natürliche Sprache)
  - E = Beziehungen zw Zielen verschiedener $v \in V$; 4 Arten:
    is_supported_by; is_undermined_by(untergraben; x wird ruch y gestört); may_be_specialised_by; must_be_considered_before(geprüft)
  - iterativer prozess; Aufstellen von high und low level goals:
    high-level: Knoten haben selten Fragmente formaler Spezi
    low-level: Knoten haben Fragmente formaler Spezi
  - Basic System Node: sollte die Basisfunktionalität des Systems beinhalten; mit Stakeholder abgeglichen werden; Ausgangspunkt für weitere Spezifikationen

Pros	Cons
Formal, Strukturiert besseres Verständnis der R durch Domain Model	R des Kunden formalisieren nicht trivial iterative Entwicklung kann inkonsistente Spezi hervorrufen

6 Design Rationale (DR)

Entwicklungsansatz das die expliziten Gründe, die zum Produkt führten, und deren Darstellung beinhaltet. Quasi welche konkrete Entscheidungen haben zum Produkt geführt? Oftmals nicht gut dokumentiert, da Entwicklungsprozess sehr komplex: bestehend aus Analyse (Problem in Subprobleme teilen) und Synthese (Teillösungen für Subprobleme zu Gesamtlösung zusammenfügen).

- Design ist fortlaufender Prozess, mit Betrachtung und Entscheidung von Alternativen; da Teilprobleme oft nicht unabhängig voneinander
- **Wicked** und Tamed Problems: schwere und klar abgegrenzte Probleme
  - jeder Lösungsversuch ändert das Problemverständnis
  - schwer bestimmbar WANN Problem gelöst, da Problem selbst nicht genau definiert werden kann
  - Stakeholder müssen sich auf hinreichend gute Lösung einigen
  - ...

Pros	cons
Aufzeichnen der Entscheidungen Kommunikation innerhalb des Projekts Integration verschiedener Stakeholder Perspektiven und Interessen Generieren von Ideen Anzeigen ungelöster/wiederkehrender Probleme	

Voraussetzung
Stakeholder sind bereit für Argumentationsprozess und rationale Entscheidungen zu treffen. Rationale Entscheidungen verkörpern oft fehlendes Wissen.

6.1 Design Space Analysis and QOC Notation - ein DR Ansatz

DSA: Verstehen eines Software Produkts; Systematische Exploration und Auswertung des Entwurfsraum mit Ziel sich für eine Alternative als Lösung zu entscheiden.
Herausforderung für Devs - Aufstellen der QOC Notation zur Unterstützung des DSA

- Ask **Q**uestions: zur Strukturierung/Vergleich von Optionen und Erstellen neuer Optionen; Option = Lösung/Antwort auf Problem/Frage
- create **O**ptions; Beispiel konkrete Eigenschaft des Produkts
- List **C**riteria zur Bewertung von Optionen und schlussendlich auch zu deren Auswahl Characteristics:

  - Messen der 'Eigenschaften' eines Kriteriums
  - muss beurteilbar sein
  - Kann sich auf (allgemeinere) Criteria beziehen
  - muss unabhängig von anderen Criteria in derselben Hierarchieebene sein
- Arguments: unterstützen die Bewertung der Optionen (Bsp: Argument ist Grund dafür das Option 2 besser als Option 1 bewertet wurde)

Questions and Options erschaffen eine hierarchische Struktur - den Entwurfsraum -> liefern **QOC Diagram**
- während den Meeting zur Diskussionslenkung benutzbar, indem Teilnehmer an Ideen der DSA orientieren
- nach Meeting zur strukturierten Protokollierung
QOC dient als verständliche Darstellung für verschiedene Stakeholder
Grobe QOC: Ideen des QOC Ansatzes nutzen für die fortlaufende Diskussion
Gründliche QIC : Protokollierung; Transkribieren; Analyse des Protokolls; QOC Diagram

Fazit:
Notation einfach genug um für alle verständlich zu sein; Flexibel genug um verschiedene Perspektiven zu repräsentieren; genau genug um Annahmen anderer zu dokumentieren
QOC Diagramme nutzen für Probleme, die keine beste oder offensichtliche Lösung haben

7 Scenario-Based Analysis and Design

Fokus auf Beschreibung wie Personen das System nutzen um eine Aufgabe zu erledigen. Organisationen können sich aus zwei Sichten betrachten: Klassifikation von Szenarios

Explizite Ansicht	**Implizite Ansicht**
Aufgaben	Wissen
Hierarchieposition	Kontaktnetzwerk
Workflow	Arbeitspraktiken
Teams	Communities

- Problemszenarios als Oberpunkt; Unterteilung in:
- Aktivitätenszenarios: typische oder kritische Dienste, die von Stakeholder erwartet werden
- Informationsszenarios: Daten die zw Nutzer und System ausgetauscht werden
- Interaktionsszenarios: Interaktion zw Nutzer und System

Übersicht der Analyse

1. Root Concept of the system to be developed
   intiale Analyse der R; Auflistung der Schlüsselaspekte der Vision; dient als Ausgangspunkt für weitere Analysen
   Dient zur Entwicklung von Scenarios
2. Field studies: Arbeitsplatz beobachten; Interviews; Artefakte bestimmen; Beziehungen zwischen Akteuren
3. Summaries:
4. Bestimmen von
   - Problemszenarios: Darstellung des Ergebnisses der Field Studies -> Schreiben von Szenarios
     Ziel: Vorstellen von Themen und Beziehungen für die weitere Entwicklung
   - Analyse von Ansprüchen (Claims): finden von Kernfeatures/Artefakten
     Ziel: Themen und implizite Beziehungen explizit herausarbeiten und für Diskussion nutzbar machen

   eng miteinander verdrahtet

Elemente eines Szenarios

- Actor
- Umgebung

- Ziele
- Pläne
- Events
- Aktionen

Zusammenfassung

- **Szenario**: Beschreibt Informationen über Akteure, ihre Annahmen über das Umfeld, ihre Ziele, Motive, Aktionen... etc
  Darstellung in verschiedenen Formen und Medien
  Informell, semi-formal, formale Notationen mgl
- **Claim/Anspruch**: Beschreibung eines Artefakts, das in Scenario vorkommt und Einfluss auf Akteure hat
  stellt implizite Beziehungen expliziter dar.
- Handlungsorientiert: Analyse der aktuellen Praxis und daraus neue Aktivitäten entwickeln
- erleichtern Nachdenken über komplexe Systeme

8 Non-Functional Requirements Framework

NFRs treiben den ganzen Entwurfsprozess voran bzw als zentraler Bestandteil des Prozesses.
NFRs stellen die Rechtfertigung für Design Entscheidungen und Bedingungen dar wie geforderte Funktionalitäten umgesetzt werden. Behandelt als **Softgoals**

- Ziel, das nicht zu 100%, aber gut genug erfüllt werden muss
- 1 Softgoal erfüllt oder lehnt/verweigert anderes softgoal ab.

Bestandteile/Ablauf des Frameworks

- Erhalten von Wissen über: Anwendungsdomain; functional R für Teile des Systems;
- Identifizieren und Zerlegen von einzelnen NFRs
- Identify Entwurfsalternativen für NFRs
- Auswerten/Behandeln von Trade Off; Prioritäten; Mehrdeutigkeiten
- Auswahl von Entwurfsalternativen
- Entscheidungen mit Design Rationale begründen
- Auswertung der Entscheidungen und deren Einfluss

9 Product Lines and Feature Models

Product Line = Serie von verschiedenen Produkten, die eine Gruppe formen und vom selben Hersteller stammen
Feature Model = Formalisierung von R; abstrahiert von konkreter Implementierung
Software Product Line = Menge von Ausprägungen eines Softwareprodukts, die auf derselben Plattform basieren; Jede Ausprägung besitzt zusätzliche, unterschiedliche Features die bestimmte Märkte/Kunden bedienen (Bsp: Windows 7 Produktlinie -> unterschiedliche Versionen)

Feature

- vom Benutzer sichtbare Aspekte/Charakteristiken des Systems
- Systemeigenschaft, die relevant für Stakeholder ist
- logische Einheit von Verhalten, das durch eine Menge von funktionalen und qualitativen R spezifiziert ist
- Feature f = (W; R; S)m mit f erfüllt Requirement R; W sind die Annahmen über das Umfeld; S = Spezifikation von f
- Stück einer Programmfunktionalität

Feature Diagram
Baum von Features; Wurzelknoten = Konzeptgedanke
Knoten = verbindliches oder optionales Feature
Feature Model = Feature Diagram + zusätzliche Daten, wie Featurebeschreibung; Prioriäten; Stakeholder;
helfen R von Produktlinien zu erkennen
Feature Interaction
Interaktion zwischen Features zeigt sich nur, wenn diese zusammengesetzt werden und nicht individuell laufen; Beispiel:
Telekomm. Call Waiting and Call Forwarding; beide Features können nicht in Verbindung aktiviert werden, denn wenn eines eine Priorität besitzt, wird das anderes disabled

Feature Oriented Software Development:
Feature Modeling; Feature Interaction und Feature Implementation sind Bestandteile
Features stehen an erster Stelle und bilden Grundbaustein für Entwicklung
Ablauf

1. Domain Analysis
2. Domain Design and Specification
3. Domain Implementation
4. Product Conifig and Generation

10 Requirements Specification and Security Risk Management

Security Concerns/Goals:

- Vertraulichkeit (erreichbar durch Kryptographie)
  nicht mehr gewährleistet, wenn vertrauliche Informationen von unautorisierten veröffentlicht werden (Bsp: Private Key verloren)
- Integrität (erreichbar durch Kryptographie)
  = Verhinderung unautorisierter Modifikation von Daten
  Datenintegrität und Quellintegrität
- Verfügbarkeit
  beinhaltet zusichern von zeitlichem und verfügbarem Zugriff auf Informationen

Zugriffskontrolle beinhaltet sowohl Authentisierung (etwas beglaubigen) als auch Authentifizierung (sich ggüber Server ausweisen)
denken wie die 'bösen Buben' (really?! irgendwie kommt mir der ganze Lehrstuhl vor wie im Kindergarten...)
Computersicherheit = Schutz der aufgebracht wird um die drei Security Concerns bei einem System zu erhalten
Authentizität = Echtheit, Überprüfbarkeit einer Eigenschaft/User
Veratwortlichkeit = Zuordnung von Aktionen zu Usern/Dateien etc

Security Bruch
3 Level

- **Low Impact**: nur geringe/begrenzte Auswirkungen..
- **Moderate Impact**: signifikante Auswirkungen..
- **High Impact**: starke/ernste Auswirkungen..

.. auf Gewinn, Individuen etc

Security Requirements - Beispiele:
Studenten - Vertraulichkeit

- Hohe Vertraulichkeit: die Noten selber - Einsicht nur von Studenten und einigen Mitarbeitern, die damit arbeiten müssen
- Moderate Vertraulichkeit: Immatrikulation
- Geringe Vertraulichkeit: Inhaltsverzeichnis = Studentenliste, Fakultätsliste; publishhed on Website etc

Patienteninformationen - Integrität

- Hohe Integrität: medizinische Unterlagen; sind sie falsch könnte eine falsche Behandlung angewandt werden
- Moderate Integrität: Website für Diskussionen von med. Themen
- Niedrige Integrität: anonyme Umfragen (zB Patientenzufriedenheit)

Verfügbarkeit - je kritischer eine Komponente, desto höher muss die Verfügbarkeit sein

- Hohe Verfügbarkeit: Authentifizierungsservice (if Service nicht erreichbar, dann restliche Funktionalität auch nicht)
- Moderate Verfügbarkeit: College Web Site
- Niedrige Verfügbarkeit: Online Phone Directory

10.1 Security Attacks

- Störung/Unterbrechen: Angriff auf Verfügbarkeit
  Traffic stören; physisch Leitungen durchtrennen
- Abhören: Angriff auf Vertraulichkeit
  Mitschneiden von Kommunikation
- Modifikation: Angriff auf Integrität
  Verändern der Daten bevor sie Ziel erreichen
- Fälschen: Angriff auf Authentizität
  Daten fälschen; sich als jemand anderes ausgeben

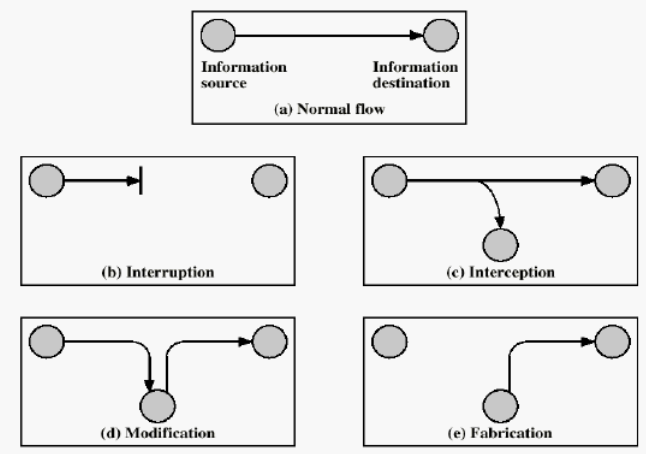


Abbildung 9: Flow bei Angriffen

10.2 Bedrohung bekämpfen

Eine mögliche Vorgehensweise:

1. Identify assets (Vermögen?)
2. Document Architecture
3. Anwendung zerlegen
4. Bedrohungen identifizieren
5. Bedrohungen dokumentieren
6. Bedrohungen bewerten

Kategorisierung von Bedrohungen - STRIDE

- **Spoofing** Angreifer mit falscher Identität erhält Zugriff
- **Manipulation** Kann Angreifer Daten manipulieren?
- **Ablehnung** Angreifer verneint Ausnutzen einer Sicherheitslücke. Kann man dies widerlegen?
- **Datenenthüllung** Kann Angreifer an private Daten kommen?
- **DoS**
- **Elevation of Privilege** Kann Angreifer privilegierten Nutzer identifizieren?

Wichtigkeit von Bedrohungen - DREAD

- **Damagepotenial** Was sind die Auswirkungen eines exploits?
- **Reproduzierbarkeit** Funktioniert der Exploit immer oder nur unter bestimmten Umständen?
- **Exploitability** Skills des Angreifers gut oder schlecht?
- **Betroffene User** Wie viele User sind betroffen?
- **Entdeckbarkeit** Wie wahrscheinlich ist die Entdeckung eines Exploits?

Risikobewältigung VS. Risikobwertung Risikobewältigungsphasen

	Risikobewältigung	**Risikobewertung**
Ziel	Risiko auf ein akzeptables Lvl reduzieren	Identifizieren und Priorisieren des Risikos
Cycle	durch alle 4 Phasen10	extra Phase dafür
Schedule	festgesetzte Aktivität	fortwährende Aktivität
Alignment	angepasst an Budget der jeweiligen Phase	nicht anwendbar

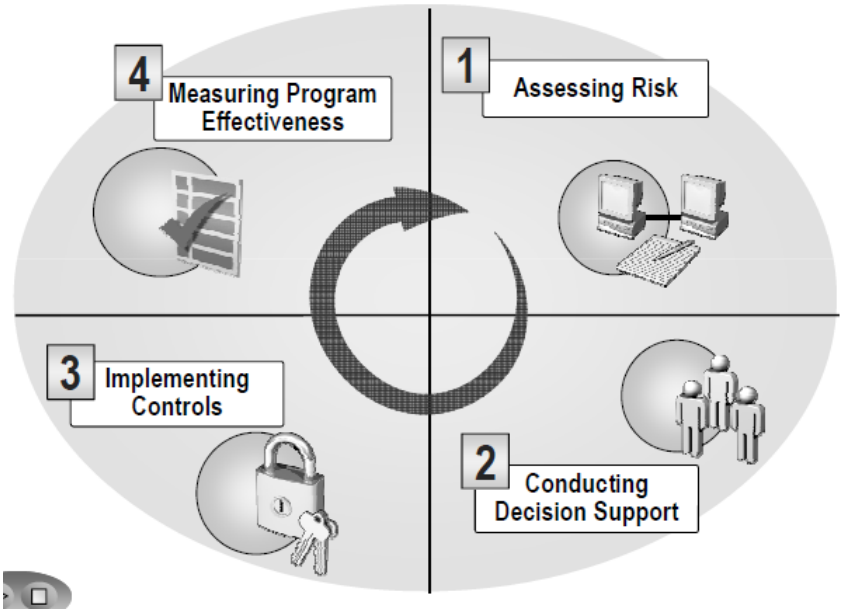


Abbildung 10: Risikobewältigung

1. Risikobewertung
   Risikodaten sammeln; Priorisieren des Risikos
2. Unterstützungen für Entscheidungen aufstellen; Quasi Risikopläne aufstellen und Kosten dafür
   a) def. functional R
   b) Identify Kontrolllösungen
   c) Abgleich Lösung gg R
   d) Grad der Risikoreduktion bewerten
   e) Kosten jeder Lösung bestimmen
   f) Strategie asuwählen
3. Kontrollmechanismen implementieren
4. Messen der Programmeffizienz

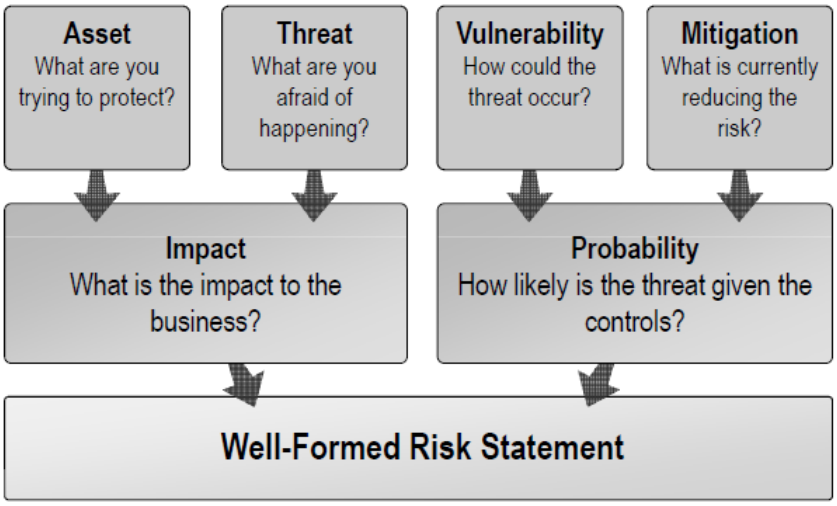


Abbildung 11: Risiko kommunizieren

10.3 Threat Modeling - Bedrohungen modellieren

Ziele

- Identifizieren wo Anwendung am verwundbarsten
- Bestimme Bedrohungen, die eine Abschwächung benötigen (= die bekämpft werden müssen?)
- Reduziere Risiko auf akzeptables Lvl durch Abschwächung

Kurzer Thread Modeling Process

1. Bestimmte Kernszenarios
2. Modelliere Anwendung mit Data Flow Diagrams
3. Bestimmte Bedrohungen für jedes DFD
4. Berechne Risiko
5. Plane Abschwächung

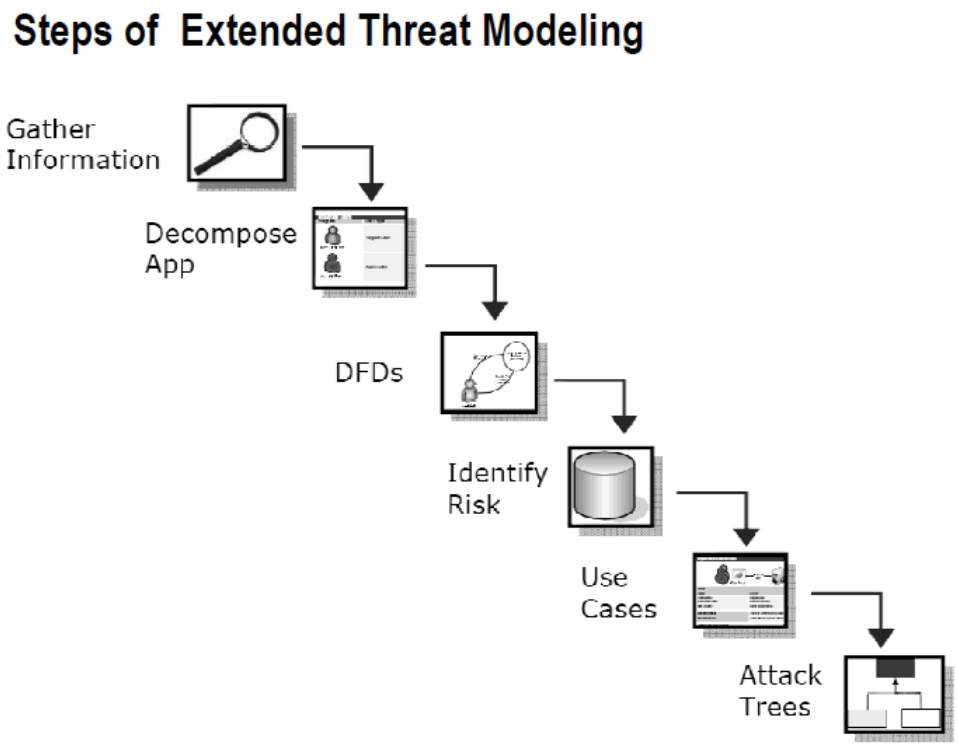


Abbildung 12: Erweitertes Thread Modeling

11 Designing Participatively - Teilnehmenderweise Entwurf

Zwei Ansätze zur Arbeitsgestaltung:

- **Job Bereicherung (Enrichment)**: Verbessern der Beziehung von Arbeiter und seiner Arbeit
- **soziotechnische Methode**
  - verbessern der Beziehung zw Gruppe und ihrer Arbeit
  - logische Analyse der technischen Komponenten des Arbeitssystems
  - Identify Systemabweichungen

First Task of Design Group: Identification of Work System Problems

- Daten über Probleme, die die Effizienz der Gesamtarbeit beeinträchtigen
- Daten über Bedürfnisse für Jobzufriedenheit und über Dinge, die das aktuelle System zusätzlich dafür benötigt
- Daten wie das neue System die Effizienz und Zufriedenheit verbessern kann

Problem solving processes

1. Problem verstehen durch Analyse von vielen, vielen Daten
2. große Datenmengen durcharbeiten und aussortieren/Auswahl was relevant ist um das Problem zu lösen
3. kleinere Datenmenge für Problemlösung nutzen

Strategies for Design process

1. 'Ideal' Systemansatz
2. Baukastenansatz

Zentrale Aufgabe des RE ist die Einbettung des neuen Systems in die bestehende Arbeitssituation/Anwendungskontext.
Relevante Zielstellungen:

- Unternehmenssicht
  mehr Effizienz in Ablauf- und Aufbauorganisation
- Arbeitswissenschaftlich
  Steigerung der Arbeitseffektivität
  Optimierung der psychophysischen Beanspruchung der Arbeiter
  Stabilisierung der Gesundheit
- Softwaretechnische Sicht
  Entwicklung von aufgabenangemessenen Softwaresystemen

Humane Factors/Ergonomics (HFE)
Wissenschaftliche Disziplin, die sich mit der Interaktion zw Menschen und anderen Elementen eines Systems und dem Beruf, der Theorie, Daten und Methoden befasst mit dem Ziel das Wohlbefinden des Menschen und die gesamte Systemperformance zu verbessern.
zugehörige Ansätze

- Ganzeinheitlicher Ansatz: Betrachte System und Menschen
- entwurfsgetrieben: Entwurfsmethoden beinhalten teilnehmende Ansätze -> Einbeziehen der Stakeholder
- Fokus auf zwei verbundene Ergebnisse: Wohlbefinden und Performance
  geringes Wohlbefinden verschlechtert die Performance und vice versa

12 Frameworks and Processes in RE

Es gibt keinen idealen RE Prozess. RE beinhaltet zwei Hauptschritte

- R Spezifikation
  Elicitation
  Analyse
  Dokumentation
  Validierung
- R Management
  Release Management
  Modification, Change Management
  Verfolgbarkeit

Ein paar relevante Aspekte in RE Prozessen

- linear oder inkrementell
  Linear: R sind eindeutig/klar; geringes Risiko; Kurze Entwicklungszeit
  Inkrementell: R nicht offensichtlich; hohes Risiko; lange Entwicklungszeit
- Rolle des Spezifikationsdokumentes (Vertrag? Müssen R von Dritten impl. werden?)
- Mglkeit des teilnehmenden Ansatzes (inkludieren des Kunden)?
- Produkt für speziellen Kunden oder Markt?

Process Patter

- **Vorschreibend**
  - alle R müssen erfüllt werden
  - Projektziele wichtiger als Deadlines und Kosten
  - SRS als Vertrag
  - oft linearer Dev-Prozess
  - oft Arbeit mit Dritten
- **Explorativ**
  - R müssen innerhalb eines Menge von globalen Projektzielen erarbeitet werden
  - Kunde/Nutzer aktiv in Prozess eingebunden
  - R priorisiert
  - oft Deadlines/Kosten wichtiger als Erreichen der Ziele
  - in der Regel inkrementell
- **Reactive**

  - System mit Standardsoftware impl
  - R müssen an diese Std-Software adaptiert werden
  - R oft nicht ins Detail ausgearbeitet
- **Kundenspezifisch**
  - initiiert durch Kundenauftrag
  - Stakeholder können identifiziert werden
  - Stakeholder auf Kundenseite sind Hauptquellen für R
- **Marktorientiert**
  - Kunden und potenzielle Nutzer sind schwer zu identifizieren
  - R meist von Devs spezifiziert; Devs müssen quasi Bedürfnisse der Kunden erahnen

Typische RE-Prozesse können durch eine Kombination von Patterns beschrieben werden

- **Vertrag**
  oft linear; vorschreibend; Kundenspezifisch
- **teilnehmender Ansatz**
  inkrementell; explorativ; kundenspezifisch
- **Productorientiert**
  inkrementell; oft explorativ; marktorientiert
- **Nutzen von Standardsoftware**
  linear od inkrementell; reactive; kundenspezifisch

12.1 User-centered Approach

interaktive Software sollte sein: **nützlich, nutzbar** und **used**(=brauchbar?!)
User-Centered Design erfordert:

- früher Fokus auf User, Aufgaben und Umfeld
- aktive Einbringung des Users
- angemessene Zuordnung von Funktionen zw User und System
- Einarbeitung von User-Feedback in das System
- iterativer Entwurfsprozess mit Entwurf, Testen und Modifizieren von Prototypen

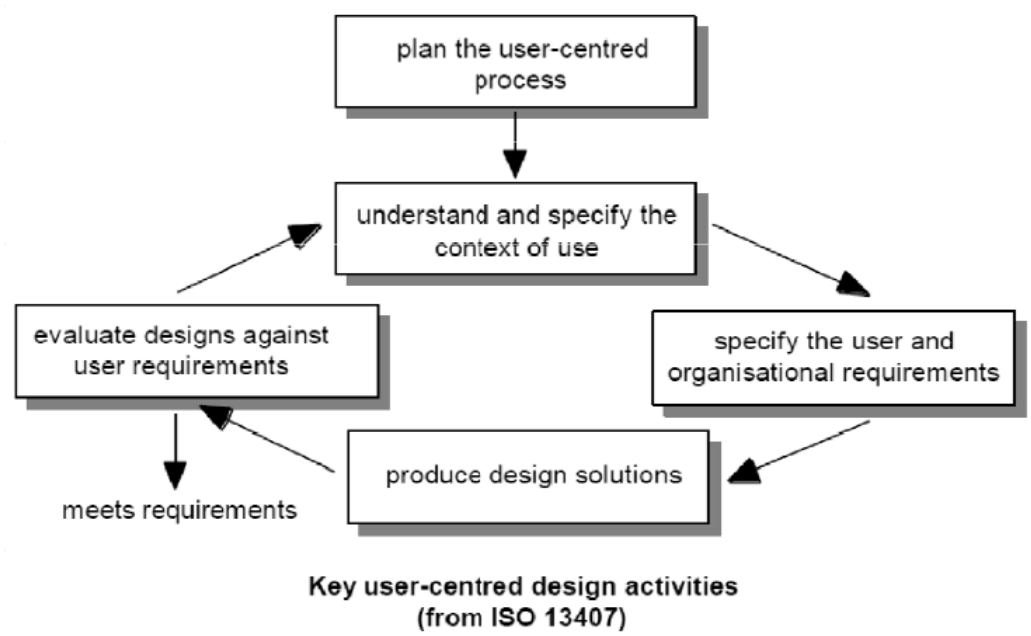


Abbildung 13: User-centered Design Activities

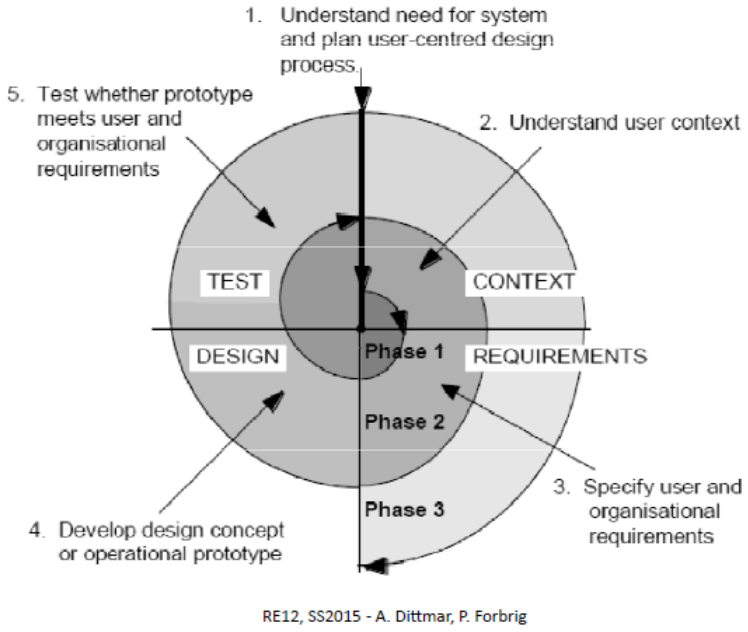


Abbildung 14: User-Centered Design - R and design cycle

13 Prototyping

- Prototyp = konkreter, aber nur teilhafte Darstellung oder Implementierung eines Systems
- verschiedene Techniken; Gemeinsamkeit = alle machen abstrakte Ideen lebendig
- interaktiv
- nützlich, weil Kunde kann sich eine Vorstellung vom System machen, mit Prototypen arbeiten

13.1 Throw Away Prototype

- entwickelt für initiale R, aber nicht genutzt fürs finale Projekt
- geschriebene Spezifikation der R
- einige Devs glauben dies ist Zeitverschwendung
- easy to use Sprache

13.2 Evolutionary Prototype

- fundamentalste Form des Prototyping
- Hauptkonzept: Robusten Prototypen entwickeln und stetig verbessern
- Ziel: User ein funktionierendes System liefern

Pros	**Cons**
ständig nach neuen Wegen zur Systemverbesserung suchen	benutzbar um Dokumentation der R zu vermeiden
erhöhte Chance Kunden zufrieden zustellen	Management wird vor.
kann benutzt werden, wenn R nicht def. sind	Long Term Wartung kann teuer sein
schnellere Auslieferung des Systems	unklare Entwurfsideen
	Informationsverlust durch viele Verbesserungen/Iterationen

Tabelle 1: Pros & Cons of evolutionary prototyping

13.3 Lo-Fi Prototyping

- Prototyp mit eingeschränkter Funktion, Interaktion
- Demonstriert allgemeines Look and Feel; stellt Konzepte, Alternativen und einfach Screen Layouts dar

- entwickelt zum Zeigen, Kommunizieren, Inspirieren; nicht als konkrete Basis zum Implementieren gedacht
- am Anfang des Entwicklungszyklus benutzt um allgemeine Konzepte zu zeigen; ohne viel Zeit in Entwicklung zu investieren
- Beispiel: Paper-Prototype

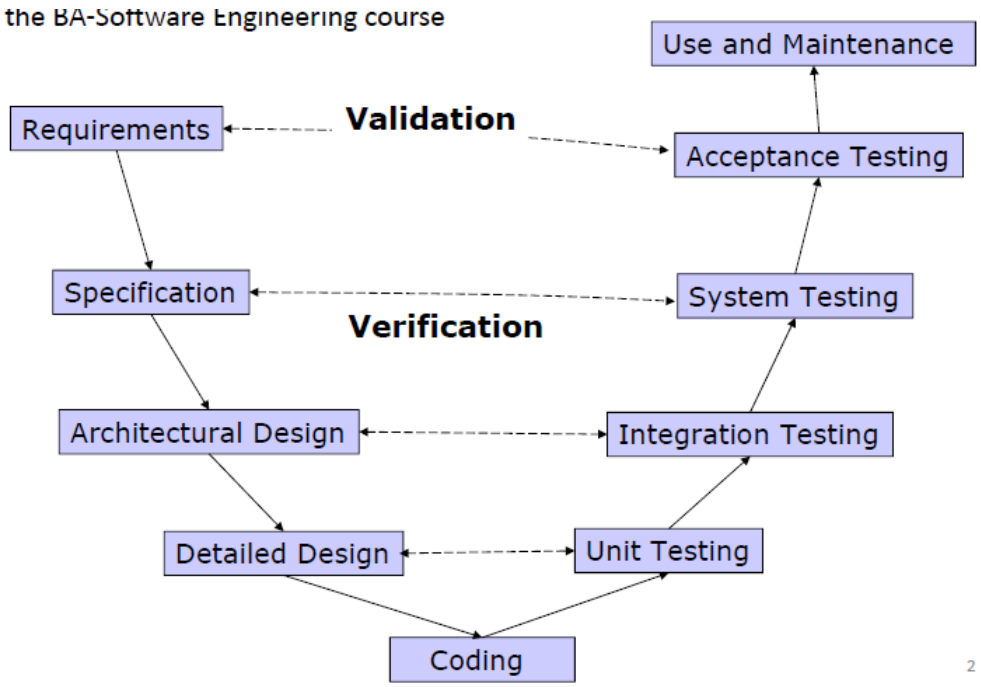
13.4 Hi-Fi Prototyping

- stellt Kernfeatures/Kernfunktionen des UI dar
- voll interaktive Systeme; User kann Daten eingeben, erhält Rückmeldungen etc; kann damit interagieren und arbeiten
- Trade-Off: Speed <-> Genauigkeit
- hohe(r) Kosten und Ressourcenverbrauch

13.5 Summary

Pros	**Cons**
Stakeholder können Möglichkeiten und Einschränkungen des Systems besser verstehen	schwer alternative Prototypen zu evaluieren
User-Centered Lösungen (betreffend das UI)	Bewertung der Stakeholder können widersprüchlich sein
Kommunikation zw User, Dev, Analyst etc	quick and dirty development
Stakeholder können einfacher neue Ideen vorschlagen	komplexe Probleme sind schwer darzustellen

14 Requirements Validation



Validation	Verifikation
Bauen wir das richtige System?	Bauen wir das System richtig?
Matcht die Problembeschreibung dem realen Problem?	Entspricht Entwurf der Spezi?
Alle Bedürfnisse der Stakeholder erfasst?	Entspricht Implementierung der Spezi?
	Macht das fertige System auch genau das was wir sagen?
	Sind die R Models untereinander konsistent?

Domain Properties: Dinge in der Anwendungsdomäne die wahr sind
Requirements: Dinge in der Anwendungsdomäne die wahr 'gemacht werden sollen' RIP german..
Specification: Beschreibung des Verhaltens damit Programm die R erfüllen kann

Zwei Verifikationskriterien
- Programm, das auf Computer läuft, erfüllt die Spezifikation
- Spezifikation erfüllt die Requirements, bezogen auf die Domain Properties

Zwei Validierungskriterien
- Haben wir alle wichtigen R aufgedeckt und auch verstanden?

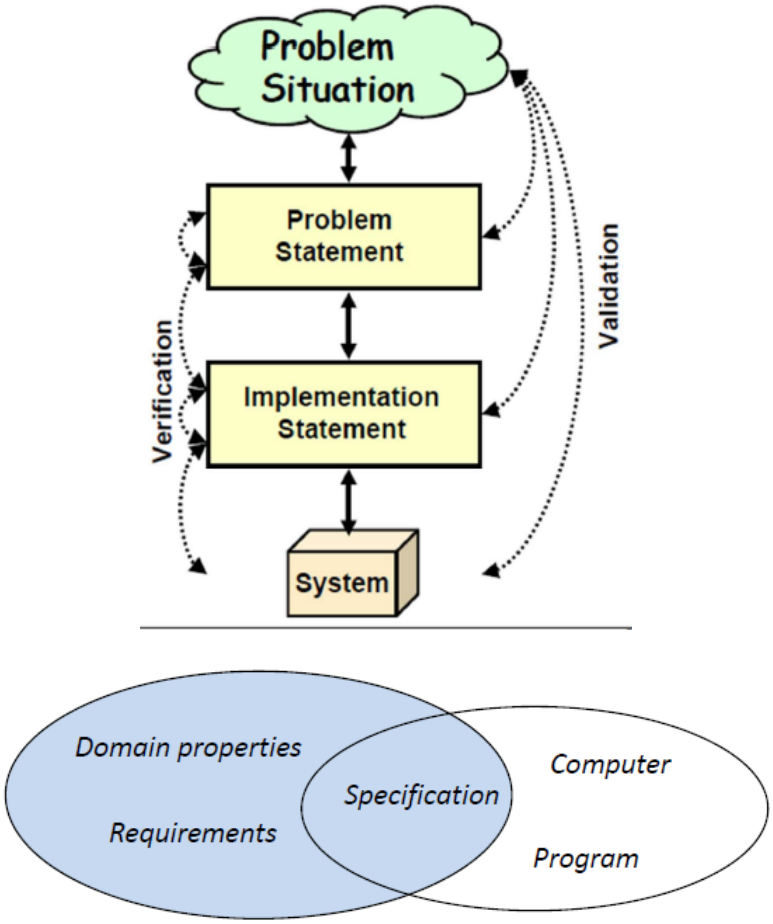


Abbildung 15: Andere Darstellung der 7

- Haben wir alle wichtigen Domain Properties aufgedeckt und auch verstanden?

Requirements Validation ist schwer, weil a) (philosophischer Natur) betrifft die Frage was ist wahr und erkennbar (knowable) und
b) (sozialer Natur) betrifft die Schwierigkeit einen Konsens zw Stakeholdern mit unterschiedlichen Zielen/Interessen zu finden.

14.1 Validation Techniken

- Review
  - Walkthrough
    verschiedene Rollen
    * Review Leader
      Führt das Meeting; sorgt für fertige Vorbereitung ; erstellt Ergebnisbericht

    * Recorder
      hält auftretende Issues fest
    * Reader (meist der Autor)
      präsentiert Produkt Schritt für Schritt
    * other reviewers
  - Inspection
    * durchgeführt von Reviewern (und nicht dem Autor)
    * Vorbereitung: Reviewer inspizieren zunächst jeder für sich
    * Collection Meeting: Reviewer treffen sich und gleichen Fehlerlisten ab
    * nutzbare Rollenverteilung
      Round Robin: rundherum - Reviewer an der Reihe > vorstellen eines Issues
      Speed Review: jeder bekommt 3 Minuten zum darstellen seiner Ergebnisse
    * benutze Checkliste mit Fragen/Issues

- Modellanalyse
  Modellanimation
  Begründung für Konsequenzen bestimmter R
  Begründung für Effekte von mgl Änderungen
- Meetings + regelmäßige Treffen mit Stakeholdern
- Prototyping

Characteristics	Throwaway prototyping	Evolutionary prototyping	Conventional development
Development approach	Quick and dirty, sloppy	Rigorous, not sloppy	Rigorous, not sloppy
What is built	Poorly understood parts	Well-understood parts first	Entire system
Design drivers	Development time	Ability to modify easily	Depends on project
Goal	Verify poorly understood requirements and then throw away	Uncover unknown requirements and then evolve	Satisfy all requirements

14.2 A Constructive Approach for Design Space Exploration

- Design Rationale: QOC Notation

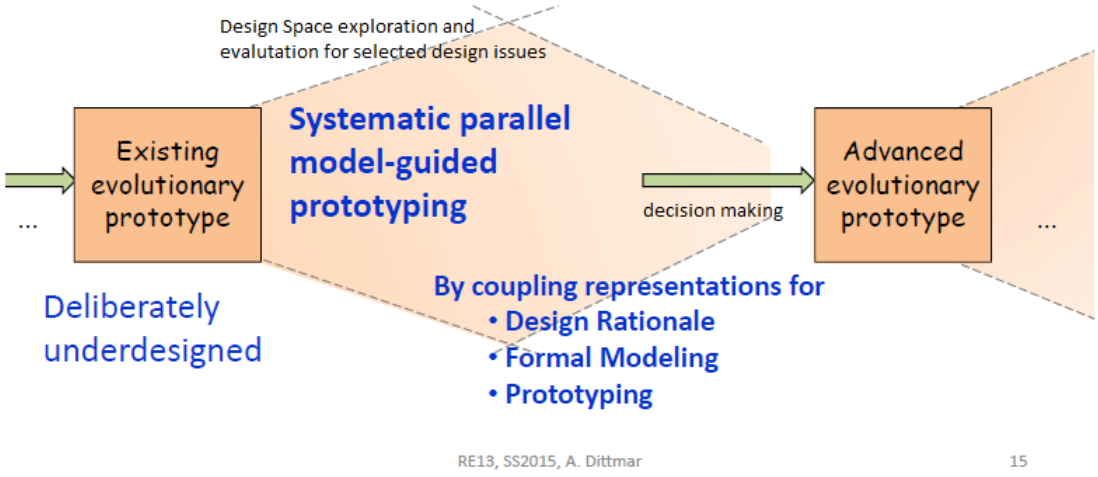


Abbildung 16: A Constructive Approach for Design Space Exploration

- Formal Modeling: HOPS Formalismen (Higher-order Processes Specification)
  - HOPS Modelle verbinden QOC Diagramme und modellgetriebene Erweiterungen des existierenden evolutionären Prototyps
    mgl, weil: HOPS Prozesse können zusammengesetzt werden; können auf Java Klassen gemappt werden; HOPS Modelle können von HOPS Tools animiert werden
  - einheitliche Beschreibung von Verhalten und strukturierten Aspekten eines interaktiven Teilsystems
  - Essentielle Modeling Konzepte:
    Prozesse; Operationen; Teilprozesse; Komponenten
  - Verteilte Beschreibung der Kontrolle
  - Mapping von HOPS Prozessen auf Java Klassen
  - Tool support

  Q-Models: HOPS Models kontrollieren Apskete des Prototypen, die sich auf offene Fragen (Q) beziehen
  O-Models: HOPS Models für Optionen (O) um alternative Lösungen zu spezifizieren und modellgetriebenes Prototyping zu ermöglichen
  C-Models: HOPS Models für Kriterien (C)

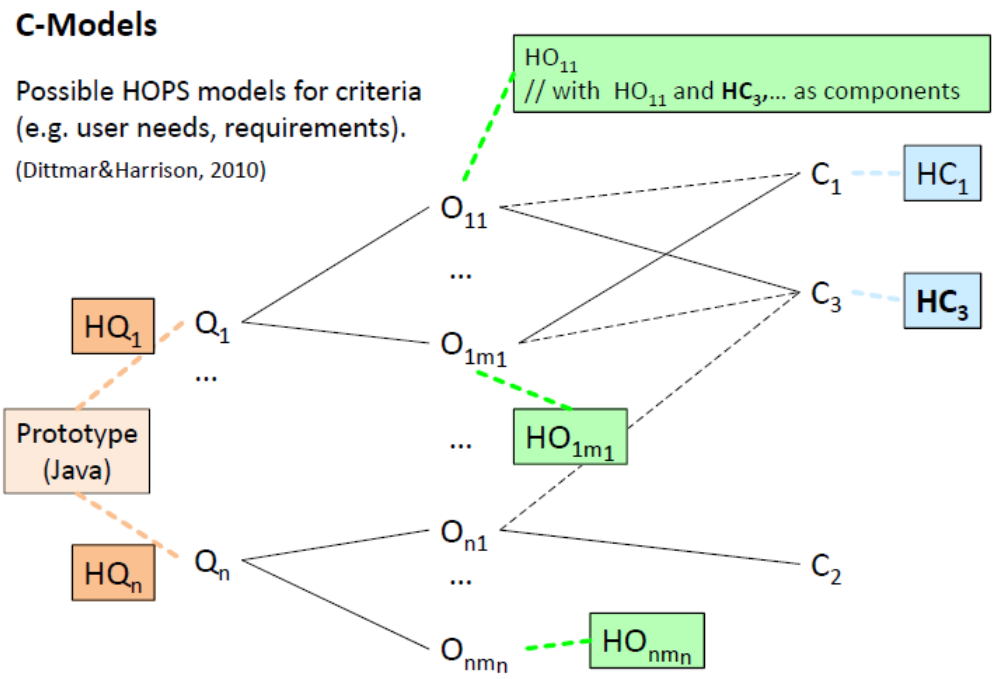
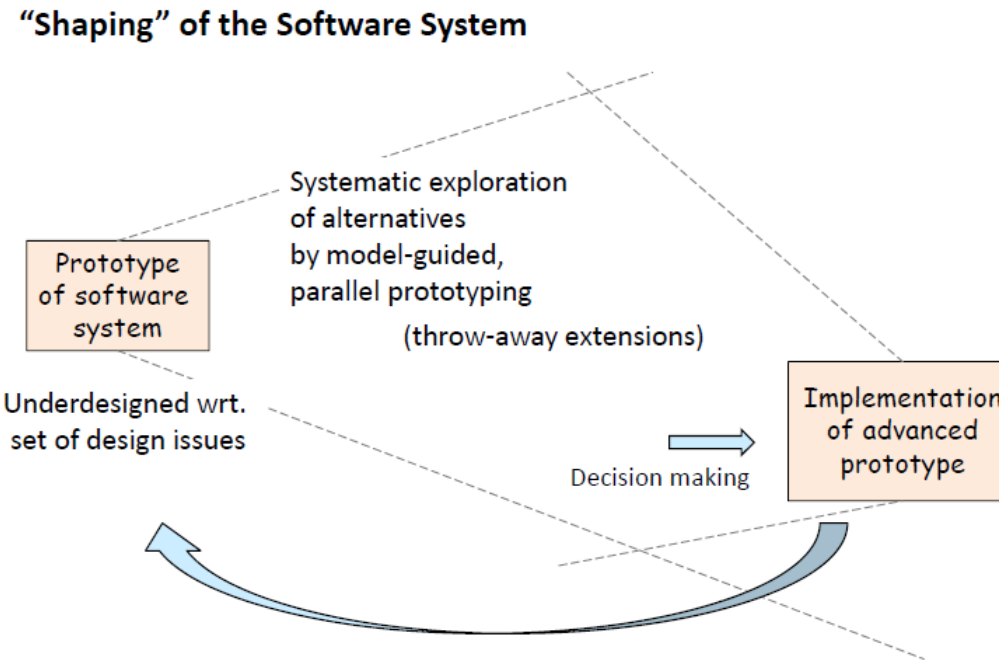


Abbildung 17: Nutzen von HOPS in QOC

- Prototyping: Java
- Systematic Parallel Model-Guided Prototyping
  Plan für Zusammenführung von QOC-Modellen, HOPS-Modellen uind Implementierungen (JAVA)



14.2.1 Summary

- Softwaresystem entwickelt sich Stück für Stück; Integration von evolutionärem und explorativem Prototyping
- Constructive Design Space Exploration: Verknüpfen von analytischen, empirischen und Implementierungs-Aktivitäten durch Evolution der Prototypen, formale Modelle

14.3 Management of Requirements

Program Types

- **S-Programs** (specificable)
  Problem kann formal in einer Menge von Spezifikation beschrieben werden; verständliche Lösungen; geringe Änderungen notwendig
- **P-Programs** (problem solving)
  iterativer Prozess notwendig um Problem zu beschreiben und eine gute Lösung zu finden
  Software entwickelt sich iterativ, weil Lösung verbessert wird oder real world sich ändert
- **E-Programs** (embedded)
  System modelliert reald-world Prozesse und wird Teil der Welt, das modelliert wird.

Systeme sind evolutionär.

Management of R

- Angemessenes Erarbeiten der R
  einzelne R sollten identifizierbar sein
  R Attribute zuweisen:
  - Identifier
  - Quelle
  - Autor
  - Creation Date
  - Änderungsdatum
  - Status (set, agreed, released, rejected...)
  - Priorität

  use of tools

- Priorisieren der R (must be/should be/nice-to-have)
  wie wichtig ist jede R für den Kunden?
  Kosten für die Impl?
  wie risikobehaftet ist der Versuch diese R zu bauen?

- Managing R Change
  Software Evolution ist unausweichlich
  Verfolgbarkeit von R ist wichtig
  - Backward Verfolgbarkeit: jede R referenziert auf vorherige Versionen/Quellen
    Rückwärts nachvollziehbar in allen Entwicklungsschritten
  - Forward Verfolgbarkeit: Vor. jede R muss eindeutig identifizierbar sein (uniqueID)
    "Forwärts"nachvollziehbar für alle neuen Dokumente die in SRS auftauchen können

  zwei Ansätze damit lar zu kommen
  - Inkrementelle Entwicklung
    kurze Entwicklungszyklen
    stabile R in einem Zyklus erreichen
    neue R in nächstem Zyklus abarbeiten
  - Explizites Änderungsmanagement
    Configuration Management (R Version)
    Baselines (stabile Version des Dokuments oder Systems)
    Formaler Bestätigungsprozess für Änderungen in nächster Baseline