

REQ - Requirements

Hinweis:

Diese Druckversion der Lerneinheit stellt aufgrund der Beschaffenheit des Mediums eine im Funktionsumfang stark eingeschränkte Variante des Lernmaterials dar. Um alle Funktionen, insbesondere Animationen und Interaktionen, nutzen zu können, benötigen Sie die On- oder Offlineversion. Die Inhalte sind urheberrechtlich geschützt.
©2016 Beuth Hochschule für Technik Berlin

REQ - Requirements



Lernziele und Überblick

Requirements Engineering (RE) ist ein Fachgebiet, welches sich mit der Anforderungserhebung beschäftigt - das heißt, mit den Anforderungen an das zu erstellende Softwaresystem.



Gliederung

Diese Lerneinheit enthält die folgenden Kapitel:

- Einführung
- Probleme und Aufgaben des REs
- Beispiele und Formalien
- Requirements Engineering II
- RE Aufgaben für Sie zur Übung
- Abschließend werden Sie einen Einblick in die Priorisierung von Aufgaben sowie das hierbei verwendbare „Eisenhower-Schema“ erhalten.



Lernziele

Ziel dieser Lerneinheit ist es, Sie mit dem Requirements Engineering vertraut zu machen, alle Eigenschaften zu kennen, Vorgehen zu erlernen und selbst eine Vorstellung davon zu bekommen, wie man als Projektleiter mit Requirements umgeht. Jeder Student sollte selbst einmal Requirements mit einem Tool erfasst haben und vom Mentor oder Projekt Feedback erhalten haben.

Nach dem Durcharbeiten sollten Sie wissen warum Requirements Engineering sinnvoll ist. Sie werden die Hauptprobleme der Softwareentwicklung und der Analyse sowie die wichtigsten Anforderungsmerkmale kennen. Das Problem der sprachlichen Unschärfe sowie der Grad der Verbindlichkeit soll Ihnen bewusst werden.



Zeitbedarf und Umfang

Für die Bearbeitung dieser Lerneinheit benötigen Sie 120 Minuten. Für die Übung benötigen Sie nochmals 270 Minuten.



Hinweis

Literatur

- **CHRIS RUPP** [Ru07]
Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis (2007)
- **KLAUS POHL** [Po07]
Requirements Engineering. Grundlagen, Prinzipien, Techniken (2007)
- **CHRISTOF EBERT** [Eb05]
Systematisches Requirements Management (2005)

Werkzeuge

- Ein Dokument / ein Excel Sheet (evtl. besser als nichts)
- „Mein eigenes RE-Tool“
- Trend Analyst der Firma GEBIT (integriert in Eclipse, mehr Use-Case Diagramme)
www.gebit.de
- RE als Teil von Projektmanagementwerkzeug InStep
www.microtool.de
- Werkzeugliste von LLC
www.jiludwig.com
- Werkzeugliste von Ian Alexander
easyweb.easynet.co.uk

Links

- Arbeitsgruppe RE der GI
www.gi-ev.de



Film

Webkonferenz zur Lerneinheit REQ

© Beuth Hochschule Berlin - Dauer: 13:42 Min. - Streaming Media 21.4 MB

Die Hinweise auf klausurrelevante Teile beziehen sich möglicherweise nicht auf Ihren Kurs. Stimmen Sie sich darüber bitte mit ihrer Kursbetreuung ab.

1 Motivation und Hintergrund

Warum sollte man sich mit Requirements-Engineering beschäftigen?

Für gewöhnlich fängt man mit kleinen Projekten an. Normalerweise haben auch die meisten Softwareentwickler etwas von UML und Use-Cases gehört. Und so beginnt man - wenn es darum gehen soll Anforderungen zu definieren - ein Use-Case Diagramm zu zeichnen und es in einem Dokument abzulegen.

Später sind es dann schon größere Projekte und man stellt fest, dass diese Thematik viel komplizierter ist und es mit zeichnen so nicht geht. Typischerweise hat man dann ein so großes Projekt, dass einem dann die initialen „Use-Cases“ Bildchen reihenweise um die Ohren fliegen - und das ganze Projekt gleich mit. Spätestens an dieser Stelle wird gewiss, dass man sich mit den Requirements hätte viel früher beschäftigen müssen: viel genauer und viel umfangreicher.

Genau das ist das Anliegen dieser Lerneinheit. Als verantwortungsbewusster Softwaretechniker sollten Sie selbst ein Konzept haben, wie sie die Thematik Requirements angehen und als Projektleiter durchziehen.

Bitte werfen Sie einen Blick auf das folgende Bild:

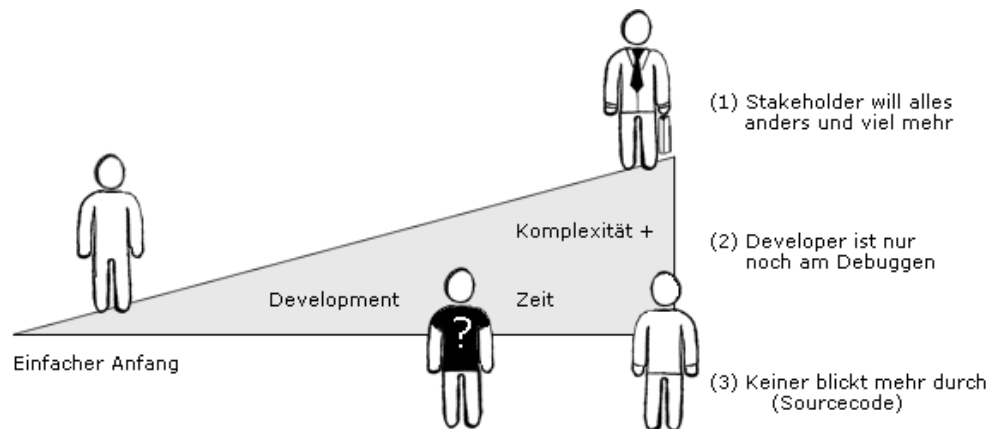


Abb.: Drei Problembereiche der Softwareentwicklung

Problembereiche

Aller Anfang ist einfach. Man kann definieren, loslegen und die Klassen wachsen und alles ist eine wunderschöne Spielwiese mit noch nicht so viel Bezug zur Realität.

Mit der Zeit werden sich - bei ernsthafteren oder größeren Projekten - jedoch drei Probleme einstellen:

1. Der Stakeholder (Auftraggeber) möchte mehr Features, er möchte das Entwickelte anders. Falls hier nicht regelmäßig eine Kommunikation stattfindet, ist das normalerweise eine schockartige Überraschung für das Team.

Der Untergang der Vasa (Siehe Anhang)

2. Der Quellcode wird viel komplexer. Die Fehlersuche wird immer schwieriger. Es gibt zu viele Abhängigkeiten. Das Hochfahren von Servern und Debuggen selbst dauert zu viel Zeit. Die Developer arbeiten dann weder qualitativ / produktiv noch kreativ.
3. Der Sourcecode wird so komplex, dass meistens selbst die Architekten oder Chefentwickler ihn kaum noch verstehen. Zudem gibt es oftmals auch noch keine dokumentierte Architektur, Architekturrichtlinien oder Mechanismen zur Begegnung von Komplexität.

Dem ersten Problem wollen wir uns in dieser Lerneinheit widmen. Die beiden anderen Probleme werden in eigenen Lerneinheiten - wie Testen oder Dependency Injection - behandelt.

Hintergrund

Requirements Engineering

Requirements Engineering besteht aus den Wörtern Requirement = Anforderung und Engineering, d. h. der ingenieurwissenschaftlichen Herangehensweise an die Anforderungsthematik. Die Anforderungen sind die Wünsche / Leistungen, die eine bestimmte Personengruppe (in der Regel der Auftraggeber) an das zu entwickelnde Softwaresystem stellt oder die es erfüllen muss. Man spricht in diesem Zusammenhang davon, Anforderungen zu erheben.

Überschneidung

Diese Thematik überschneidet sich mit der generellen Analysetätigkeit im Softwarelebenszyklus. So gibt es:

- allgemeine (Informatik) Anforderungen
- die konkrete Software Requirements Spezifikation
- Teile des (Pflichten- und besonders des) Lastenheftes müssen Requirements enthalten
- und das Anforderungsmanagement (also RM alias Requirements Management)

Im Laufe dieser Lerneinheit werden alle diese Gebiete näher betrachtet.

Anfangsphase

Es gilt wie so häufig, Fehler frühzeitig zu vermeiden daher ist auch die Anfangsphase eines Projektes sehr wichtig. Fehler zu korrigieren, die bei der initialen Weichenstellung gemacht werden kosten später viel mehr Geld und Zeit. Dies wird zu Beginn oft verdrängt oder vergessen. Untersuchungen zufolge sind 65% aller groben Fehler in Softwareprojekten auf Fehler in der Analysephase zurückzuführen. Siehe dazu auch die Statistiken in der Lerneinheit: *Einführung in die Softwaretechnik* (Seite 6)

Anforderungsdefinition

Anforderungsdefinitionen werden leider viel zu oft in irgendwelchen Word-Dokumenten beim Projektleiter abgespeichert oder „vergammelt“ in einigen Use-Case Diagrammen. Damit dies nicht geschieht, gehört ein RE Konzept in das gesamte Projekt integriert. Also die:

- korrekte
- vollständige
- qualitativ hochwertige

Erhebung und Verwaltung von Anforderungsmerkmalen.

Oft werden fehlende Requirements im Projekt durch Vermutungen ersetzt. Weiterhin gibt es oft den Fall, dass der Auftraggeber seine Meinung (oder damit auch seine Anforderungen) ändert. Nicht selten haben viele Mitarbeiter des Auftraggebers stark unterschiedliche Vorstellungen davon, was das System leisten soll.

Ziel

Ziel sollte es also auch sein, ein Verfahren zu etablieren, in dem Requirements verwaltet werden und nachvollziehbar dokumentiert sind.

2 Aufgaben und Probleme

Vorteile guter Requirements

Sehr gute Requirements haben vielfältige Vorteile für das Projekt. Sie dienen dazu, in der frühen Phase - Stichwort Pflichtenheft / Lastenheft / Ausschreibung / Vertrag - klare Zielvorstellungen zu fixieren und Vorstellungen besser zu konkretisieren. Die Requirements selbst haben unmittelbaren und starken Einfluss auf das Design und die Architektur des Systems. Und schließlich wird von RE-Experten auch eingeworfen, dass gute fixierte Requirements die Kommunikation aller Beteiligten am Projekt verbessern. Mit einem Gesamtverständnis für Requirements werden Fehler, schlechte Vermutungen und falsche Annahmen vermieden.

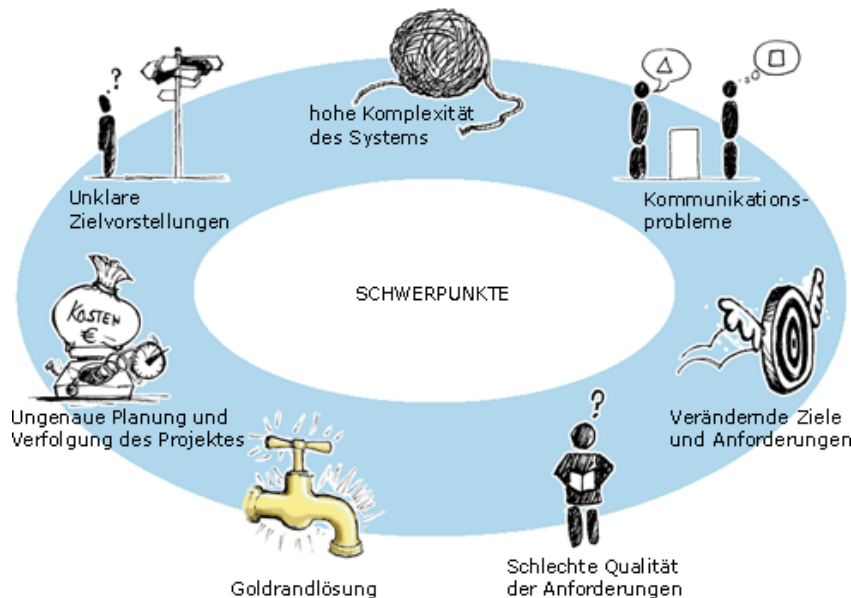


Abb.: Sieben wesentliche Hauptprobleme

Nach **RUPP** [Ru07] gibt es sieben wesentliche Hauptprobleme, die bei der Analyse des zukünftigen Systems eine Rolle spielen:

Unklare Zielvorstellungen

Oft unterscheidet sich die letztlich anwendende Personengruppe von den Personen, die das System in Auftrag gegeben haben. Die Anforderungen aller Repräsentanten (Stakeholdern) zu sammeln und unter einen sinnvollen Hut zu bringen ist zwar schwierig, aber wichtig.

Hohe Komplexität des Systems

Heutige Softwaresysteme werden immer komplexer. Es treten immer mehr Wechselwirkungen und Abhängigkeiten auf, wodurch die Berücksichtigung dieser immer schwieriger wird.

Kommunikationsprobleme

Alle am Projekt beteiligten haben einen verschiedenen Wissenshintergrund und auch verschiedene Wünsche. Dazu kommt noch eine verteilte Produktentwicklung in verschiedenen Sprachen.

Verändernde Ziele und Anforderungen

Diese sind als moving targets oder creeping requirements bekannt. Leider oder zum Glück verändern sowohl die Menschen, als auch das Projekt sich im Laufe der Zeit. Ein gutes Requirements Management kann hier helfen, die wirklich wichtigen Änderungen zu erfassen, zu diskutieren und adäquat zu berücksichtigen. Ohne ein gutes RE kommt es immer wieder vor, dass es sich Entscheidungsträger einfach mal nach Lust-und-Laune anders überlegen. Und ein System nach Lust-und-Laune zu reimplementieren ist Gift für jedes Projekt.

Schlechte Qualität der Anforderungen

Hier sind nach **RUPP** die folgenden Probleme in den Anforderungsbeschreibungen zu finden: Mehrdeutigkeiten, Redundanzen, Widersprüche und ungenaue Angaben. Die Auswirkungen dieser Probleme in den Requirements sind leicht vorzustellen.

Goldrandlösungen (Gold-plating)

Im XP-Sprachgebrauch ist hier auch oft von *goldenen Wasserhähnen* die Rede. Damit sind Features gemeint, die meistens von begeisterten Entwicklern hinzugefügt werden, die aber nicht wirklich notwendig und relevant sind.

Ungenaue Planung und Verfolgung des Projektes

Basierend auf ungenauen Anforderungen werden Projekte zu optimistisch geplant und geraten dann zu teuer und in der Entwicklungszeit zu lang. War beispielsweise Performance bisher keine Anforderung; wird es dies aber gegen Ende des Projektes, so zeigt sich, dass - z. B. von der Architektur - ein ganz anderes System gebaut hätte werden müssen.

2.1 Anforderungsmerkmale

Anforderungen

Nach der Sichtung der Literatur und einiger Werkzeuge zeigt sich, dass Anforderungen idealerweise alle diese Merkmale erfüllen müssten:

- korrekt
- vollständig
- eindeutig definiert / abgegrenzt
- verständlich beschrieben
- atomar
- identifizierbar
- einheitlich dokumentiert
- notwendig
- nachprüfbar / testbar
- rück- und vorwärts verfolgbar
- konsistent
- klassifizierbar (jur.)
- Gültigkeit / Aktualität
- realisierbar
- bewertbar / priorisierbar

Punkt „klassifizierbar“

Besonders beim Punkt „klassifizierbar“ weist RUPP darauf hin, dass man rechtlich bindende Requirements markieren / gruppieren können muss. Man stelle sich vor, dass würde es bei der Entwicklung von Bankensoftware nicht geben...

Bei der Überlegung, welches RE-Werkzeug man verwendet oder sogar selbst baut, könnte man daher sehr gut diese Kriterienliste zu Rate ziehen.

Hier noch ein Beispiel wie das in Form von Karten notiert werden könnte.



Rolloverbild

Kriterien auf Karten vermerken		
Eindeutige ID	Art der Anforderung	Auflistung der Voraussetzungen für die Anforderung
Requirement #:	Requirement Type:	Event/use case #'s:
Description: Erläuterung des Schwerpunktes der Anforderung in einem Satz		
Rationale: Begründung der Notwendigkeit der Anforderung		
Originator: Urheber der Anforderung	Fit Criterion: Umsetzungsfähigkeit der Anforderung	
		Andere Anforderungen, die nicht mit dieser Anforderung vereinbar sind.
Customer Satisfaction:	Customers Dissatisfaction:	
Priority: Priorität des Kunden	Conflicts:	
Supporting Materials:	Verweis zu den Dokumenten mit der Beschreibung der Anforderung	
History: Entwicklung, Veränderungen, Überarbeitungen, etc.		
Grad der Zustimmung für die Verwirklichung der Anforderung. Von 1 = uninteressiert bis 5 = sehr erfreut		Grad der Unzufriedenheit bei nicht Verwirklichung der Anforderung. Von 1 = uninteressiert bis 5 = sehr unzufrieden

Die Erfassung von Requirements sollte ausführlich sein, um spätere Missverständnisse zu vermeiden. Dies bezieht sich nicht nur auf den sprachlichen Aspekt, sondern auch auf die Eigenschaften, die Requirements besitzen können.

Dies könnten beispielsweise die folgenden Attribute sein:



Beispiel

Attribute von Requirements
<pre> <liste> ID Datum Author Kurzbeschreibung lange Beschreibung Referenz auf den Use Case Referenz auf sonstige Dokumente Wer nimmt das Requirement ab? Wie lautet das Abnahmekriterium? Priorität Konflikte Abhängigkeiten juristische Relevanz sonstige Anmerkungen Historie </liste> </pre>

Überlegen Sie, welche Attribute Sie für Ihre Requirements als wichtig erachten und erfassen Sie diese entsprechend genau.

2.2 Sprachliche Unschärfe

Beispiel von O'Reilly

Hier ein anschauliches Beispiel aus Head First Software Development von O'Reilly:

Tom Says: „*The customer should be able to search for trails*“
(Wanderruten)




- The customer should see a map of the world and then enter an adress to search for trails near a particular location.
- The customer should be able to scroll through a list of tourist spots and find trails that lead to and from those spots
- The customer should be able to enter a ZIP code and a level of difficulty and find all trails that match that difficulty and are near that ZIP

Sie erkennen die Problematik?

Sprache ist oft so unendlich unscharf. Ziel muss es also sein, die Requirements gemäß der vorigen Kriterienliste so klar wie möglich zu definieren.

2.3 Grad der Verbindlichkeit

Verschiedene Grade
der Verbindlichkeit

Wie man aus den Anforderungen an ein  Schachprogramm sehen kann, gibt es verschiedene Grade der Verbindlichkeit. Diese sollten auch sprachlich präzise definiert werden. Der Stakeholder sollte wissen, dass ein zu schwacher Begriff vielleicht zur Folge hat, dass das Feature nicht realisiert wird. Ein zu starker Begriff kann unter Umständen aber auch dazu führen, dass das System viel teurer wird als geplant.

Abstufungen

In der Literatur finden sich daher die folgenden Abstufungen zum Grad der Verbindlichkeit wieder:

- **muss** (Pflicht)

Diese Anforderung ist also Verbindlich. Oft wird auch **soll** verwendet, was aber nicht so gut ist. Das Wort „soll“ hat eine unschärfere Komponente in sich und ist nicht so verpflichtend, wie es in einem Requirements Dokument sein sollte. Im englischen wird in Requirements Dokumenten „shall“ verwendet. Das Englische „must“ ist zwar noch härter, widerspricht aber der englischen Höflichkeit.

- **soll / sollte** (Wunsch)

Die Anforderung ist nicht maximal zwingend, aber doch sehr stark erwünscht. Gerade bei solchen Kategorie Requirements kommt später das Erwachen, dass es nicht finanzierbar ist. Z. B. soll das System performant sein oder möglichst 1000 Anfragen pro Sekunde verarbeiten können. Im Englischen wird hier „should“ verwendet.

- **wird** (Absicht)

Man hat vor, ein „wird“ Requirement zu realisieren. Ob das dann auch erreicht wird, ist zwar wahrscheinlich aber nicht 100 % sicher. Im Englischen wird hier „will“ verwendet.

- **kann** (Vorschlag)

Eine Anforderung kann realisiert werden. Muss aber nicht. Im Englischen hier: „can“.

Die Anforderungen „wird“ und „kann“ werden etwas seltener verwendet. Natürlich ist ein Wunsch und ein Vorschlag irgendwie ähnlich und oft sogar gleichbedeutend. Sorgen Sie also dafür, dass Ihre Dokumente und alle Beteiligten wissen, welche Wortwahl welche Auswirkungen hat.

Oft legen Projektleiter daher die folgende Struktur fest.

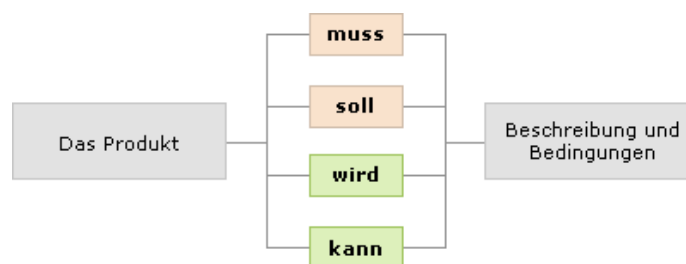


Abb.: Hierarchische Struktur der
Verbindlichkeit

Diese Struktur wird sogar von manchen Programmen unterstützt, vorgegeben oder geprüft.

2.4 Priorisierung

Schema der
Priorisierung

Wichtig und schwierig ist das Priorisieren von Aufgaben. Die meisten Programme geben hier schon ein Schema vor.

Erfahrungsgemäß sind drei bis fünf Stufen völlig ausreichend, um zu priorisieren. Also „sehr hoch“, „hoch“, „mittel“, „gering“, „null“. Diese sollten dann nummeriert oder mit Ziffern versehen (4 bis 0 oder A bis D) und auf jeden Fall farblich unterlegt werden.

Ähnlich der ABC Analyse ist es hier interessant, sich alle Anforderungen auch noch einmal unter dem Gesichtspunkt des Eisenhower-Schemas anzusehen. (Tipp: suchen sie mal bitte in dem Web nach „Eisenhower Schema“ und ggf. auch nach „ABC Analyse“)

Tasks

Die folgende Grafik zeigt, was Sie mit Tasks machen könnten:

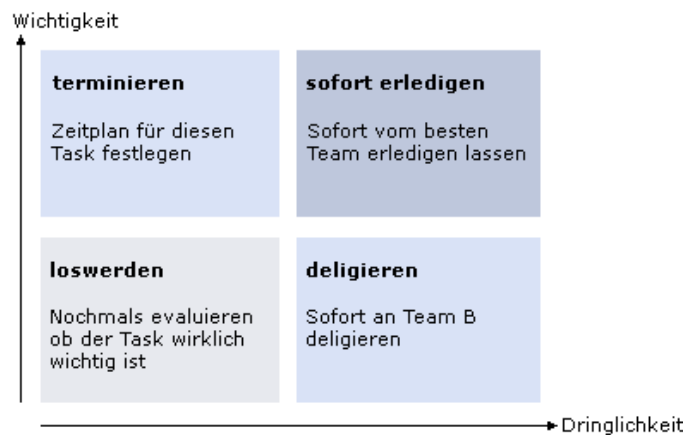


Abb.: Eisenhower Schema

Schema für Führungskräfte

Die Y-Achse ist dabei die **Wichtigkeit** und die X-Achse die **Dringlichkeit**.

Das Schema ist zwar besonders für Führungskräfte geeignet, die delegieren können, aber es lohnt immer auch, Anforderungen und konkrete Tasks oder User-Stories daraufhin abzuklopfen.

3 Beispiele und Formalien



Beispiel

Formale Anforderung an ein Schachprogramm

1. Bei der Entwicklung **sollen** alle Best Practices angewendet werden.
2. Das Schachprogramm **muss** Arena kompatibel sein
3. Programmiersprachen:
 - Der Kern **muss** schnell sein: ➡ Java
 - Der Wrapper code **sollte** fit für SE-Tools sein ➡ Java
 - Umliegender Code **kann** auch in Ruby sein.
4. Das Schachprogramm **soll** eine dreistufige Eröffnungsbibliothek haben:
 - Perfekte DB auf Basis von FEN + N*[Move gewichtet mit %]
 - EdIDB auf Basis von ECO (sollte von Ruby kompiliert werden / regexp)
 - bigbook.txt
5. Das Schachprogramm **muss** einen schnellen Bitboard-Kern haben



Was ist hier von
Punkt 1 zu halten?

Im Beispiel wurde eine sprachliche Darstellung gewählt. Es liegt aber an Ihnen abzuwägen, wann eine sprachliche Beschreibung und wann eine graphische Darstellung bspw. mittels Use-Cases geeignet ist.

Die im Beispiel genannten Requirements sind noch nicht perfekt:

- Üblicherweise sollten in der ersten Version noch keine technischen Details enthalten sein
- Sie sollten in der ersten Version auch immer aus der Sicht des Anwenders geschrieben sein.

3.1 Atlantic Systems Guild

Strukturierung von Anforderungen

Wie wir eben gesehen haben, gibt es sowohl intrinsische Anforderungen, als auch Anforderungen vom Endanwender. Sie sollten sich also genau überlegen welche Kategorie von Anforderungen Sie erfassen.

Die wohl bekannteste Strukturierung von Anforderungen stammt von der „Gang of Six“, der Atlantic Systems Guild (www.systemsguild.com). Von dieser Gruppe von Autoren stammt das Template für Anforderungen. Das Original finden Sie im Internet unter:

<http://www.volare.co.uk/template.htm>

Eine der Übungen am Ende dieser Lerneinheit, wird die Erkundung dieses Templates sein.

Anforderungen

Wir listen diese Anforderungen aus obiger Referenz noch einmal auf:

FUNCTIONAL REQUIREMENTS:

- The Scope of the Work
- The Scope of the Product
- Functional and Data Requirements

NON-FUNCTIONAL REQUIREMENTS:

- Look and Feel
- Usability and Humanity
- Performance
- Operational
- Maintainability and Support
- Security
- Cultural and Political
- Legal

Weiterhin werden noch Project Drivers, Project Constraints und Projekt Issues gelistet, welche teilweise auch in die Analyse, Design oder das Projektmanagement fallen und dort behandelt werden.

Wie wird mit Templates umgegangen?

Die genannten Anforderungen können als eine Art zusätzliche Checkliste verwendet werden, um nach Requirements zu suchen. Die meisten Requirements sind sicherlich funktional und beschreiben das Produkt, denn daran denkt man bei der Requirements Analyse sicherlich zuerst. Häufig werden die nicht-funktionalen Requirements vergessen. Daher kann es nicht schaden, zu den nicht-funktionalen Requirements konkrete Requirements zu entwickeln und diese im System zu notieren.

Tools

Der Markt der Werkzeuge für Requirements-Engineering ist sehr gespalten. In der Praxis wird man feststellen, dass es grob drei Gruppen gibt:

1. zu einfache Tools (Word- oder Excel-Dokument),
2. nicht passende Tools (GEBIT - etwas zu Use-Case lastig)
3. zu komplexe, riesige Tools (InStep von MicroTool oder das RUP Tool), die eher für Großprojekte geeignet sind.

Daher muss jeder Requirements-Engineer meist alle Werkzeuge selbst evaluieren bis er ein Passendes findet oder sich sogar selbst eines programmieren. Es lohnt sich jedoch, diese Zeit zu investieren.

3.2 Was noch zu beachten ist

Neben den schon genannten Hinweisen zur Erstellung von Requirements sollten Sie noch einige weitere Dinge beachten.

- Machen Sie sich Gedanken wie Sie die Anforderungen dokumentieren. Welches Tool oder welche Dokumentenart. Das ist wichtig.
- Wie viele Stakeholder gibt es? Sind diese verfügbar und können sie kommunizieren?
- Selbst die Höhe des Budgets hat großen Einfluss auf die Anforderungen und die Anforderungsermittlung.
- Versuchen Sie bei der Ermittlung der Anfragen immer zu erspüren, ob es implizite Anforderungen / Fakten gibt.
- Einige Anforderungen sind den Stakeholdern manchmal gar nicht bekannt. Versuchen Sie also unbewusste Wünsche „hellzusehen“.

Kreativitätstechniken

Natürlich müssen Sie auch die Palette der Kreativitätstechniken kennen, um am besten an die Anforderungen heranzukommen. Dies gilt freilich auch dann, wenn Sie das existierende System nur kennenlernen möchten oder in der Analyse / Designphase mehr herausbekommen möchten:

- Brainstorming
- Mind-Mapping
- Befragen
- Beobachten
- Aufschreiben
- Interviewen
- Video / Audioaufzeichnungen

Zusammenfassung

In dieser Lerneinheit wurde Folgendes vorgestellt:

- Requirements Engineering überschneidet sich mit der generellen Analysetätigkeit im Softwarelebenszyklus.
- Ein RE Konzept - also die korrekte, vollständige und qualitativ hochwertige Erhebung und Verwaltung von Anforderungsmerkmalen gehört in das gesamte Projekt integriert.
- Requirements haben unmittelbaren und starken Einfluss auf das Design und die Architektur des Systems.
- Requirements dienen dazu, in der frühen Phase klare Zielvorstellungen zu fixieren und Vorstellungen besser zu konkretisieren.
- Sieben wesentliche Hauptprobleme, die bei der Analyse des zukünftigen Systems eine Rolle spielen, können beschrieben werden.
- Der Grad der Verbindlichkeit lässt sich in vier Abstufungen darstellen. Pflicht, Wunsch, Absicht und Vorschlag.
- Drei bis fünf Stufen sind für das Priorisieren völlig ausreichend.
- In der ersten Version sollten Anforderungen immer aus der Sicht des Anwenders verfasst werden und noch keine technischen Details enthalten.

Sie sind am Ende dieser Lerneinheit angelangt. Auf den beiden folgenden Seiten finden Sie noch die Übungen zur Wissensüberprüfung und Aufgaben zum Selbststudium.

Wissensüberprüfung



Multiple Choice

Übung REQ-01

Hauptprobleme

Nach CHRIS RUPP gibt es sieben Hauptprobleme, die bei der Analyse des zukünftigen Systems eine Rolle spielen. Ordnen Sie die Kurzbeschreibungen dem Problem zu.

A	Die letztlich anwendende Personengruppe unterscheidet sich oft vom Auftraggeber.	<input type="checkbox"/>	Goldrandlösungen
B	Schwierigkeiten, auftretende Wechselwirkungen und Abhängigkeiten zu berücksichtigen.	<input type="checkbox"/>	Kommunikationsprobleme
C	Unterschiedlicher Wissenshintergrund der am Projekt beteiligten.	<input type="checkbox"/>	Ungenaue Planung und Verfolgung des Projektes
D	Auch als moving targets oder creeping requirements bekannt.	<input type="checkbox"/>	Hohe Komplexität des Systems
E	Mehrdeutigkeiten, Redundanzen, Widersprüche und ungenaue Angaben.	<input type="checkbox"/>	Verändernde Ziele und Anforderungen
F	Features, die nicht wirklich nötig und relevant sind	<input type="checkbox"/>	Schlechte Qualität der Anforderungen
G	Projekte geraten dadurch zu teuer und die Entwicklungszeit zu lang.	<input type="checkbox"/>	Unklare Zielvorstellungen

? Test wiederholen Test auswerten



Formulieren

Übung REQ-02

Problembereiche

Bei der Entwicklung größerer Projekte können sich schnell Probleme ergeben. Benennen Sie die drei Problembereiche der Softwareentwicklung.

Bearbeitungsdauer: 5 Minuten

[Lösungshinweis](#)



Multiple Choice

Übung REQ-03

Grad der Verbindlichkeit

In der Planung sollte der Grad der Verbindlichkeit sprachlich möglichst präzise definiert sein. Ordnen Sie die entsprechenden Begriffe der jeweiligen Abstufung zu.

A	Pflicht	<input type="checkbox"/>	Soll(te)
B	Wunsch	<input type="checkbox"/>	Wird
C	Absicht	<input type="checkbox"/>	Kann
D	Vorschlag	<input type="checkbox"/>	Muss

? Test wiederholen Test auswerten




Formulieren

Übung REQ-04

Eisenhower Schema

Die Priorisierung von Aufgaben ist wichtig und schwierig. Ein Ansatz hierfür ist das Eisenhower Schema. Skizzieren Sie dies mit X- und Y-Achse.

Bearbeitungsdauer: 15 Minuten

 [Lösungshinweis](#)



Recherieren

Übung REQ-05

Selbststudium zum Thema Requirements

Aufgaben zum Selbststudium, die garantiert zur Thematik RE fit machen:

1. **Recherchieren** Sie selbst zum Thema Requirements-Engineering.
2. Stöbern Sie dazu bitte besonders unter www.systemsguild.com bzw. unter www.volere.co.uk/template.htm
3. Bauen Sie selbst ein kleines Requirements Engineering Tool.
Selbst für den Anfang einen kleinen Prototypen zu bauen ist nicht schwer. (So z. B. ein GUI mit [NetBeans](#) Matisse gezeichnet welches eine Liste zeigt und verwaltet und dies mit [db4o](#) gespeichert - das geht recht schnell).
4. Können sie schnell in MS-Access ein RE-System hacken?
Stufe 2: Mit Versionierung?
 1. Wie würden Sie das angehen?
 2. Was sind die Requirements für ein RE-Tool?

Bearbeitungsdauer: 120 Minuten

Projektaufgabe



Projektaufgabe

Übung REQ-06

Projektaufgabe - Requirements


Wählen Sie sich ein kleines **Entwicklungsprojekt** für das gesamte Studienmodul Softwaretechnik und verwenden Sie dieses Projekt für alle Lerneinheiten. Beispielsweise das schon genannte RE-Tool, oder ein anderes Tool was bei Softwaretechnik hilft oder ein Tool welches Sie schon immer mal bauen wollten (nennen wir es Projekt A). Im Verlauf des Moduls wenden Sie alle Themen wie Testen, Refactoring oder auch Requirements-Engineering an.

Als Aufgabe dieser Lerneinheit ermitteln Sie die Requirements für ihr Projekt und verwalten diese.

Bearbeitungsdauer: 120 Minuten

Appendix

Der Untergang der Vasa

Auszug aus dem Buch  Produktiv programmieren von NEAL FORD

Im Jahr 1625 gab König Gustav II Adolf von Schweden das stolzeste Kriegsschiff in Auftrag, das die Welt jemals gesehen hatte. Er heuerte den besten Schiffbauer an, ließ extra einen Wald mit den mächtigsten Eichen hochziehen und begann mit der Arbeit an der »Vasa«. Immer wieder stellte der König neue Anforderungen auf, die das Schiff noch großartiger machen sollten, über und über prunkvoll dekoriert. Irgendwann entschied er, dass das Schiff – anders als jedes bisher auf der Welt gebaute Schiff – mit zwei Kanonendecks bestückt sein sollte, auf dass es das stärkste Kriegsschiff auf den Weltmeeren werde. Und wegen eines gerade aufbrechenden diplomatischen Konflikts brauchte er es so schnell wie möglich. Natürlich hatte der Schiffbauer beim Entwurf des Schiffs nur ein Kanonendeck vorgesehen, aber da der König es so wollte, bekam er auch sein zweites Kanonendeck.

Da es schnell gehen musste, hatte man keine Zeit für üblichen »Lurch-Tests«, bei denen eine Gruppe von Matrosen von einer Seite des Schiffs zur anderen läuft, um sicherzugehen, dass das Schiff nicht zu stark ins Schaukeln gerät (mit anderen Worten: dass es nicht zu kopflastig ist).

Auf ihrer Jungfernfahrt sank die Vasa binnen weniger Stunden. Dadurch, dass dem Schiff all die »Features« hinzugefügt worden waren, hatte es seine Seetüchtigkeit verloren. Bis in das frühe 20. Jahrhundert lag die Vasa auf dem Grund der Nordsee, dann wurde das gut erhaltene Schiff gehoben und in ein Museum gebracht.

Und hier stellt sich eine interessante Frage: Wer war schuld daran, dass die Vasa sank? Der König, weil er ständig neue Features verlangt hatte? Oder die Schiffbauer, die das Gewünschte bauten, ohne ihre Bedenken laut genug zu äußern? Sehen Sie sich in dem Projekt um, an dem Sie gerade arbeiten: Sind Sie vielleicht gerade dabei, eine Vasa zu bauen?