Requirements Engineering

Mikael Svahnberg[1]

2016-03-16

[1] Mikael.Svahnberg@bth.se

Requirements Engineering
A Brief Overview

- Develop requirements through an iterative co-operative process of analysing the problem
- Documenting the resulting observations in a variety of representation formats
- checking the accuracy of the understanding gained

See the course `PA1410 Requirements Engineering`

Difficulties in Requirements Engineering

- The customer may not be able to express what he or she wants so that you are able to understand it
  - Tacit knowledge
- Finding the right people to ask
- Getting access to the right people
- Handling large amounts of requirements
- Specifying the requirements so that both you, the customer, your developers, and your testers can understand and use them

Requirements Engineering Phases

- Elicitation
- Analysis & Negotiation
- Validation
- Documentation
- Management

Discuss: RE Sources and Techniques

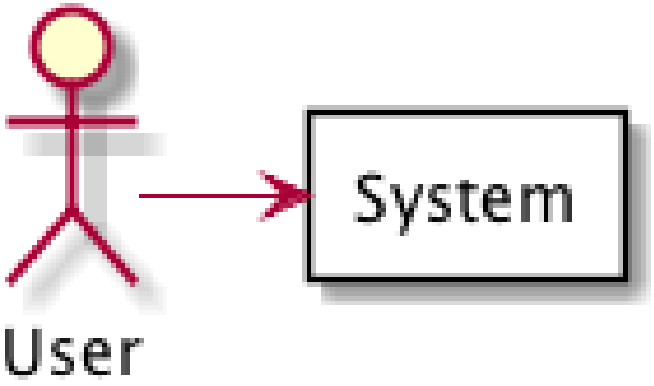
- How do we find requirements?
- Where do we find requirements?

Requirements Elicitation Techniques

- Interviews
- Use-Case-based Discussions
- Observations
- Brainstorming
- Questionnaires
- Prototyping
- Incremental Deliveries
- Analysis of Written Documents

Discuss: System Scope

- What is the scope of the system?
  - System boundaries
- What should you do?
- What should you not do?
  - Balance: Requirements, Schedule and Budget
- During Analysis / Design: Black Box vs White Box



Requirements Specification

- What the proposed system shall do
  - At what quality level
- A documented common view
- An agreement between developers and customer
  - Sign a contract based on the requirements specification
- Involve client in process
- Decrease Risk

Quality Attributes

Accessibility, accountability, accuracy, adaptability, additivity, adjustability, affordability, agility, audiability, buffer, space performance, capability, capacity, clarity, code-space performance, cohesiveness, commonality, communication cost, communication time, compataibility, completeness, component integration time, component integration cost, composability, comprehensibility, conceptuality, conciseness, confidentiality, configurability, consistency, controllability, coordination cost, coordination time, correctness, cost, coupling, customer evaluation time, customer loyalty, customizability, data-space performance, decomposability, degradation of service, dependability, development cost, development time, deversity, distributivety, domain analysis cost, domain analysis time, efficiency, elasticity, enhanceability, evolvability, execution cost, extensibility, external cosistency, fault-tolerance, feasibility, flexibility, formality, generality, guidance, hardware cost, impact analyzability, independence, informativeness, inspection cost, inspection time, integrity, internal consistency, inter-operability, intuitiveness, learnability, main-memory performance, maintainability, maintenance cost, maintenance time, maturity, mean performance, measurability, mobility, modifiability, modularity, naturalness, nomadicity, obervability, off-peak-period performance, operability, operating cost, peak-period performance, performability, performance, planning cost, planning time, plasticity, portability, precision, predictability, process management time, productivity, project stability, project tracking cost, promptness, prototyping cost, prototyping time, reconfigurability, recoverability, recovery, reengineering cost, reliability, repeatability, replacability, replicability, response time, responsiveness, retirement cost, reusability, risk analysis cost, risk analysis time, robustness, safety, scalability, secondary-storage performance, sensitivity, security, similarity, simplicity, software cost, software production time, space boundness, space performance, specificity, stability, standardizability, subjectivity, supportability, surety, survivability, susceptibility, sustainability, testability, testing time, throughput, time performance, timeliness, tolerance, tracebility, trainability, transerability, transparancy, understandability, uniform performance, uniformity, usability, user-friendliness, validity, variability, verifiability, versatility, visibility, wrappability

160 NON-FUNCTIONAL REQUIREMENTS IN SOFTWARE ENGINEERING

accessibility, adaptability, affordability, availability, capacity, cohesiveness, communication time, component integration cost, comprehensibility, confidentiality, controllability, correctness, customer evaluation time, data-space performance, dependability, distributivity, domain analysis time, enhanceability, extensibility, feasibility, generality, impact analyzability, inspection cost, inter-operability, learnability, maintenance cost, mean performance, modifiability, nomadicity, operability, performability, planning time, precision, productivity, promptness, reconfigurability, reengineering cost, replaceability, responsiveness, risk analysis cost, safety, security, simplicity, space boundedness, stability, supportability, susceptibility, testing time, timeliness, trainability, understandability, usability, variability, visibility,

accountability, additivity, agility, buffer space performance, clarity, commonality, compatibility, component integration time, conceptuality, configurability, coordination cost, cost, customer loyalty, decomposability, development cost, diversity, efficiency, evolvability, external consistency, flexibility, guidance, independence, inspection time, internal consistency, main-memory performance, maintenance time, measurability, modularity, observability, operating cost, performance, plasticity, predictability, project stability, prototyping cost, recoverability, reliability, replicability, retirement cost, risk analysis time, scalability, sensitivity, software cost, space performance, standardizability, surety, sustainability, throughput, tolerance, transferability, uniform performance, user-friendliness, verifiability, wrappability

accuracy, adjustability, auditability, capability, code-space performance, communication cost, completeness, composability, conciseness, consistency, coordination time, coupling, customizability, degradation of service, development time, domain analysis cost, elasticity, execution cost, fault-tolerance, formality, hardware cost, informativeness, integrity, intuitiveness, maintainability, maturity, mobility, naturalness, off-peak-period performance, peak-period performance, planning cost, portability, process management time, project tracking cost, prototyping time, recovery, repeatability, response time, reusability, robustness, secondary-storage performance, similarity, software production time, specificity, subjectivity, survivability, testability, time performance, traceabi lity, transparency, uniformity, validity, versatility,

Table 5.2. A list of non-functional requirements.

More Structured Quality Attributes: ISO9126

- Functionality
  - Suitability, Accuracy, Interoperability, Security, Functionality Compliance
- Reliability
  - Maturity, Fault Tolerance, Recoverability, Reliability Compliance
- Usability
  - Understandability, Learnability, Operability, Attractiveness, Usability Compliance
- Efficiency
  - Time Behaviour, Resource Utilisation, Efficiency Compliance
- Maintainability
  - Analysability, Changeability, Stability, Testability, Maintainability Compliance
- Portability
  - Adaptability, Installability, Co-Existence, Replaceability, Portability Compliance

Discuss: Requirement Attributes

- Requirements ID
- Title
- Description
- Rationale
- Restrictions & Risks
- Source
- Fit Criterion / Test Case
- Customer Priority
- Dependencies

Discuss

What is the purpose of each of these attributes?

Format of Requirements

- What the system should do
  - not how the system should do it
- Testable - Measurable
- Unambiguous
- Only one requirement
- Unique
- Understood by all parties
- Text, Figure, Diagram, Table?

User Stories

- Simpler template:
  - As a *type of user*, I want *some goal* so that *some reason*.
- Written on index cards or post-it notes.
- Often with *acceptance tests* on the flip-side.

Levels of Requirements

T. Gorschek and C. Wohlin. Requirements abstraction model. *Requirements Engineering*, 11:79–101, 2006:

- Goals : Aligned with product Strategies
- Features : High-level descriptions of system functionalities
- Functions : break-down of each feature
- Components : further breakdown

Do we see the system as a *Black Box* or a *White Box*?

Black box What can we do towards the system, how does it respond?

White box What does the system do internally?

Discuss: Good and Bad Examples

- The system should be easy to use

Discuss: Good and Bad Examples

- The system should be easy to use

ID: Req.QA.Useability
Title: Useability for New Users
Description: The system shall be easy to learn for new users.
Rationale: The average user is not accustomed to using computers.
Source: Customer Meeting 2002-01-14, PG Gyllenhammar
Value Scale: Number of Hours it takes for a novice user to learn a new operation
Wanted value: 0,5 h / operation
Worst case value: 1,5 h / new operation

Discuss: Good and Bad Examples

- The system should be stable
- The user should be able to log in. If the user fails to log in after three attempts the user account should be locked.

Customer Contacts

- Respect
- Responsibility
- Commitment to the Customer
- Credibility
- Professional
- Deliver at least what you have agreed upon
  - Deliver at most?
- Only one Customer? Only one Stakeholder?

Customer Meetings

- Be prepared
- Have an Agenda
- Document what is said
- Reply quickly after a meeting with your perception of what was said
  - e.g. in the form of a draft requirements specification
- Act professional
  - You are in control – you should act like it

Contracts

- “A written judicial document defining the terms for business related agreements”
  - Verbal agreements
  - Written agreements
- Defines
  - Deliverables
  - Payments
- Written in sunshine, used in storm
- Contract Types
  - Fixed price
  - Running price
  - Cost-plus
  - Roof price
  - Combinations

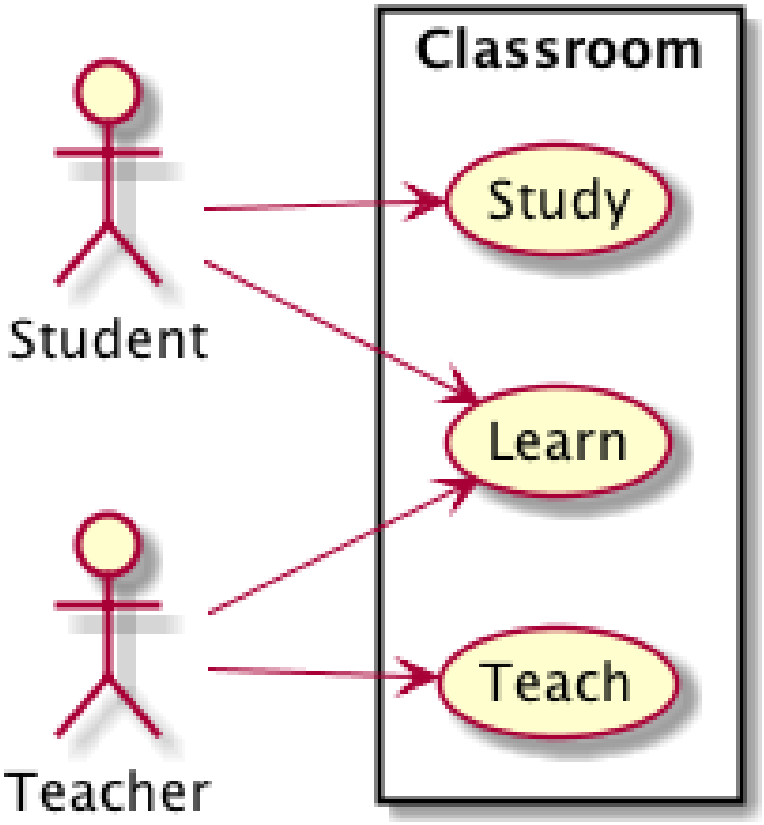
Contract Contents

- Definition of Services
- Time Period
- Contact persons
- Costs
- Deliveries
- General Conditions
- Connected to:
  - A Specific Version of the Requirements Specification
  - Project Plan?

Contract: Important Points

- The contract defines what you shall do.
- The contract also defines what you can expect from the customer.
- You sign the contract knowing that you can deliver what is specified, under the specified conditions.

Back to RUP / Use Cases



Discuss: Good and Bad Requirements II

Users shall be able to view a personal calendar and recent notifications in the system.

Use Case: View Calendar and Notifications
Actors: System Users
Description:
A user requests to view their personal calendar.
The system displays the users' personal calendar.
A user requests to view their recent notifications.
The system displays the users' recent notifications.

Discussion on Use Case Ranking

Increase ranking of a use case if it

- has direct impact on architectural design
  - example: adds classes to domain layer, require persistent services
- includes risky, time-critical, complex functions
- involves new research or technology
- represents primary business processes
- directly supports revenue or decreased costs

Discuss

For each of these cases, why does it increase the rank of a use case?

Use Case Ranking Techniques

- Scored (Numerical Weights)
- Discrete (High, Medium, Low)
- Simple Ordering (bubble sort?)
- MoSCoW (Must have, Should have, Could have, Won't have)
- Cumulative Voting