Requirements Analysis

Software Requirements and Design

CITS4401

Week 3 – Part 1

The 4 Major Activities of Requirements Engineering

- Elicitation
- **Analysis**
- Specification
- Validation

Requirements Classification

Types of Requirements (Recap)

Functional requirements describe the ***functions*** that the SW is to perform

Example: The SW shall enable a student to enroll in a class

Non-functional: requirements that ***constrain*** the solution

Performance	Maintainablity	Safety
Reliability	Security	Privacy
Interoperability		

Example: Student addresses and other personal information should not be released to any unauthorized party (*privacy*).

Examples of Non-functional Reqs

Performance: the SW will respond to client web activity in a timely and convenient way

Maintainability: the SW will be implemented using modern programming practices that maximize the maintainability and reusability of designs and code

Interoperability: the SW shall run on XX phones and YY devices

Useability: the SW shall conform to ISO 9241 usability standard [ref]

Examples of Non-functional Reqs

Safety: the SW will transition to an agreed safe state within 1 second of any sensor readings outside their thresholds

Reliability: the SW shall be available for use as much as comparable productivity tools.

Security: the SW shall protect users' personal information from XXX penetration attacks

Privacy: the SW shall protect each user account with password entry

More requirements classifications

Requirement Priority: the higher the priority the more essential the requirement is for meeting the overall goal of the SW. A fixed-point scale such as Must have, Should have, Could have, and Won't have.

Requirement Scope: The extent to which a requirement affects the SW and components. Eg. Non-functional requirements such as response times have global scope: their satisfaction can not be allocated to a single component.

Requirement Volatility: Some requirements will change during the lifetime of the SW and even during development. It is useful to estimate the likelihood that a requirement will change so that developers can consider designs that are more tolerant of change.

Viewpoints of Requirements

- **Interactor viewpoints:** people or other systems that interact directly with the system
  - e.g. users of a messaging app
- **Indirect viewpoints:** stakeholders who do not use the system, but influence requirements
  - e.g., payroll system
- **Domain viewpoints:** constraints of the domain that influence requirements
  - e.g. healthcare system

Requirements Negotiation

Conflict / Negotiation

- **Requirement Negotiation or Conflict Resolution**
- The both concern **resolving problems** with requirements where **conflicts** occur
  - between two stakeholders requiring mutually incompatible features,
  - between requirements and resources, or
  - between functional and non-functional requirements

- In most cases, it is unwise for the software engineer to make a unilateral decision.
- So it becomes necessary to **consult with the stakeholder(s)** to reach a consensus on an appropriate tradeoff.
- It is often important, for **contractual reasons**, that such decisions be **traceable** back to the customer.

Detecting Conflicts

Use Order of Priority

- **Determining importance to the customers**: Determines the degree of importance of each requirement to the customer.
- **Prioritizing due to time and resources constraints**: There may not be enough time or resources to implement all requirements, so the most critical should be implemented first.
- **Identifying conflicting requirements**: Helps to identify conflicting requirements.
- **Planning for future releases**: Can help you plan successive releases of a product by identifying which requirements should be done first, and which should be left to successive releases.

MoSCoW method

- MoSCoW stands for Must have, Should have, Could have, Won't have (two "o" are inserted at appropriate places to give the word MoSCoW).

- We can ask the client to group their requirements of the system into two lists: the DO list and the NOT DO list.

- The MoSCoW rules have an advantage over the simple ranking method – e.g., if there are many (say 1000) requirements then ranking using numbers from 1 to 1000 would be difficult, but grouping them into two lists using the MoSCow rules would be a lot easier.

Formal Methods

- Construct a ***mathematical model*** of the requirements
- Use ***logical analysis*** to verify properties and identify inconsistencies
- Most methods have ***tool support*** and some have automatic analysis
- Popular models include 1st order logic, ***set theory*** (eg. Z), ***temporal logic***, ***state machines***

Weaknesses?

- What are the weaknesses of these 3 strategies for large projects?
- The requirements are NOTtruly independent, yet all these strategies assume they are;
- The client might not know their priorities;
- Different stakeholders do not usually agree with the priorities of the requirements.

Resolving Conflicts

Why negotiation?

- Negotiation is introduced to facilitate requirements elicitation and analysis.
  - Encourages communication
  - Aids in understanding
  - Reveal conflict, solution exploration, collaborative resolution
  - Improves agreement level
  - Develop stakeholders' satisfaction
  - Improves requirements quality

Dilbert's negotiations

Negotiation for agile software development

- Negotiation for traditional software methods focuses on revealing conflicts and improving understanding of requirements.
- As agile methods focus on **involvement of customer**, whose role is to provide and prioritize new system requirements, negotiation for agile software development should therefore focus on resolving these system requirements, e.g.,
  - Can they be implemented within the time frame?
  - Can they be implemented within the budget?
  - Can these requirements be prioritized?

Negotiation for agile software development (cont.)

- Agile methods have to rely on contract where customer pays for time spent on system development rather than the time on developing a specific set of requirements.
  - Negotiate on what to be delivered, i.e., the product
  - Software developer should be **realistic** on what they can deliver (i.e., do not over-promise just to get the contract signed)

Boehm's Win-Win Spiral

Multi-stakeholder involvement with coordination and collaboration based on

i) *Win Conditions* capture the desired objectives of the individuals

ii) *Conflict/Risk/Uncertainty specifications* (*CRU's*) capture the conflicts between win conditions and their associated risks and uncertainties.

iii) *Points of Agreement* (*POA's*) capture the agreed upon set of conditions which satisfy stakeholder win conditions and also define the system objectives.

Win-Win Model

1. identify next-level stakeholders
2. identify their win conditions
3. reconcile win conditions
4. evaluate product and process alternatives; resolve risks
5. define next level of product and process
6. validate next level of product and process
7. review & commitment; return to 1

Feasibility Studies

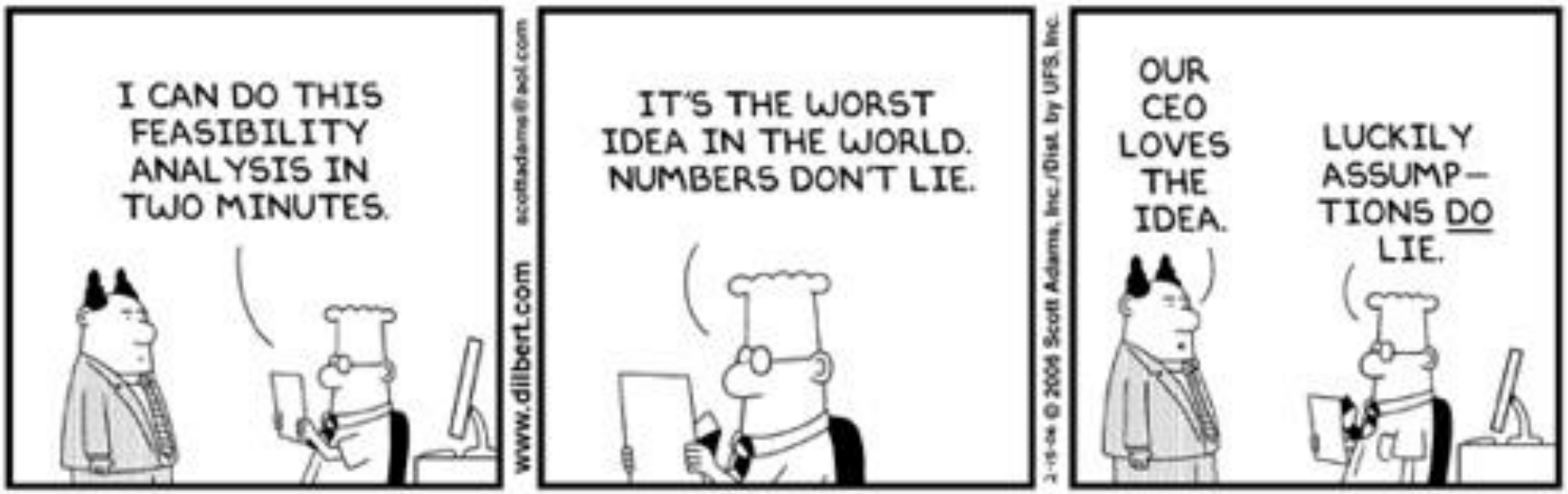
Feasibility studies

- INPUT
  - set of preliminary business requirements
  - an outline description of the system
  - how the system is intended to support business processes
- OUTPUT
  - a report recommending whether or not it is worth carrying on with the requirements engineering and system development process

More questions for the feasibility studies

- How would the organisation cope if this system were not implemented?
- What are the problems with current processes?
- How would the new system alleviate these problems?
- Does the system require technology that has not previously been used in the organisation?
- What must be supported by the system? What need *not* be supported by the system?

Dilbert's feasibility



© Scott Adams, Inc./Dist. by UFS, Inc.

Requirements Evolution

Issues

- Issues
  - requirements inevitably change.
- Definitions
  - a classification of requirements according to the types of change which may occur
- Techniques
  - SCM (Software Configuration Management), traceability tables, good record keeping, metrics

SCM = Software Configuration Management

Reasons for requirements change

- User gains better understanding of the requirements from the requirements elicitation, analysis and validation process
- New ways of working result from the introduction of the software system itself
- Changes to the *environment* of the organisation
- Changes to systems or processes *within* an organisation

- Development is a *discovery* process

Consequences of requirements change

Consequences

- Increased costs – bad, leads to fights
- Delays, schedule slip – bad, leads to fights
- Increased cost + delays – really bad, bring in the lawyers, sell your house
- Can break “customer trust” – really bad, you lose the next 5 contracts as well as this one.

Severity of the consequences depends when in the life cycle the requirements change

- Best case – review requirements specification
- Worst case – changes to requirements, design, implementation, tests and documentation

Two Classes of Requirements

- Enduring Requirements
  - Derive from an organisation's core activity
  - Relate directly to the problem domain
  - Relatively stable

- Volatile Requirements
  - Derive from the environment of the system
  - Likely to change during development or afterwards

Classes of Volatile Requirements

- Emergent
  - from improved *understanding* of the problem
- Consequential
  - as a result of *using* the delivered system
- Mutable
  - from changes to the *environment* of the organisation
- Compatibility
  - from changes to processes *within* the organisation

Traceability tables

- Uniquely number all requirements
- Identify specific aspects of the system or its environment classified by, for example,
  - *Features:* important customer observable system or product features
  - *Dependency:* how requirements are related to one another
  - *Subsystems:* governed by a requirement
  - *Interface:* relation to internal and external interfaces

Taceability Tables

A Database of Requirements

- Manage requirements as a *live* repository
- Manage traceability tables
- Record rationale (reasons for a requirement)
- Record sources (where req. comes from)
- Record rejected requirements
- Identify volatile requirements (so they can be traced later)

Recommended reading

- Pressman, *Software Engineering: A Practitioner's Approach,*
- Understanding Requirements > Establishing the Groundwork > Recognising Multiple Viewpoints/Collaboration
- Understanding Requirements > Negotiating Requirements

Note: Pressman chapter numbers differ by edition,
so I've given chapter > section

- Bruegge and Dutoit, *Object-Oriented Software Engineering – Using UML, Patterns, and Java,* 3rd ed., Prentice Hall, 2010

  Chapter 4 Requirements Elicitation