Risks Implied by Bad Requirements Specifications

- 
  - negotiation
  - requirement specification
  - design / implementation
  - quality assurance
  - acceptance
  - documentation
  - re-use

- • without specification, programmers may just “ask around” when in doubt, possibly yielding different interpretations → difficult integration
- • without specification, the user’s manual author can only describe what the system does, not what it should do (“every observation is a feature”)
- • without a description of allowed outcomes, tests are randomly searching for generic errors (like crashes) → systematic testing hardly possible
- • without specification, it is unclear at delivery time whether behaviour is an error (developer needs to fix) or correct (customer needs to accept and pay) → nasty disputes, additional effort
- • without specification, re-use needs to be based on re-reading the code → risk of unexpected changes, later re-implementations.
- • the new software may need to adhere to requirements of the old software; if not properly specified, the new software needs to be a 1:1 re-implementation of the old → additional effort.
Discovering Fundamental Errors Late Can Be Expensive

Relative error costs over latency according to investigations at IBM, etc. By (Boehm, 1979); Visualisation: Ludewig and Lichter (2013).

The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements...No other part of the work so cripples the resulting system if done wrong. F. P. Brooks (Brooks, 1995).

Topic Area Requirements Engineering: Content

• Introduction
• Vocabulary: Requirements (Analysis)
• Importance of Requirements Specifications
• Requirements Specification
• Requirements Analysis
• Desired Properties
• Kinds of Requirements
• Analysis Techniques
• Documents
• Dictionary
• Specification
• Requirements Specification Languages
• Natural Language
• Decision Tables
• Syntax, Semantics
• Completeness, Consistency, ...
• Scenarios
• User Stories, Use Cases
• Live Sequence Charts

Recall: Structure of Topic Areas

Example: Requirements Engineering Vocabulary
e.g. consistent, complete, tacit, etc.
Techniques informal semi-formal formal

Content

Requirements Specifications
It is not trivial to have both, low maintenance effort and low access effort.

- Being too specific, may limit the design decisions of the developers, which may cause unnecessary costs.

- Requirements specification documents should be as precise as possible. However, a very precise requirements specification should not introduce new unclarities or rooms for interpretation. The requirements specification should not contain information that is not relevant to the project.

- In almost all cases, requirements specification documents are not a solution but should serve as a basis on which the software development is based. To maximise confusion, we may occasionally (inconsistently) call it "feature specification" or "non-functional requirements" or "functional requirements" and most changes require reading beforehand.

- Note: One and the same content can serve both purposes; only the title defines the purpose then.

- Recall: The sources of requirements are documented, requirements are uniquely identifiable, and all requirements (existing in somebody's head, or a document, or ...) should be present.

- Therefore, the content of requirements specification documents should be:
  - desired: consistent, correct, complete, not unnecessarily complicated, easily usable, easily understandable, easily maintainable, not unnecessarily complicated, easily usable, easily understandable, easily maintainable.
  - objective: testable, non-abstract.

- Consider the following examples:

  - Precise, not realisable: "the list of participants should be sorted conveniently".
  - Precise, testable: "the list of participants should be sorted by immatriculation number, lowest number first".
  - Vague: "how are things to be done?".

- A requirements specification should always be as complete and correct as possible. For analysing an existing requirements/feature specification "finding out what the exact requirements are" is in almost all cases not a requirement but a service provided by a consultant. It is not a solution but should serve as a basis on which the software development is based. To maximise confusion, we may occasionally (inconsistently) call it "feature specification ('Pflichtenheft')", "requirements specification ('Lastenheft')", or "non-functional requirements". The source of requirements is documented, requirements are uniquely identifiable, and all requirements (existing in somebody's head, or a document, or ... ) should be present.

- In the following (unless otherwise noted), we discuss the analysis of requirements specifications. We have already seen the importance of requirements specifications, and we shall now present analysis techniques.

- Requirements on Requirements Specification Documents

  - Things which are not relevant to the project should not be constrained.
  - A requirements specification does not constrain the realisation more than necessary.
  - A requirements specification does not introduce new unclarities or rooms for interpretation.
  - All requirements (existing in somebody's head, or a document, or ...) should be present.
  - A requirements specification is a technical document, the equivalent of a natural language text.
  - A requirements specification is not a specification in the sense of a dictionary or a definition of a term.

- Requirements Specifications should be:
  - Precise, objective, testable, non-abstract.
  - Precise, objective, testable, abstract.
  - Precise, objective, testable, not precise, abstract.

- Content of a requirements specification should be:
  - desired: consistent, correct, complete, not unnecessarily complicated, easily usable, easily understandable, easily maintainable.
  - objective: testable, non-abstract.

- Analysing an existing requirements/feature specification: "finding out what the exact requirements are" is in almost all cases not a requirement but a service provided by a consultant. It is not a solution but should serve as a basis on which the software development is based. To maximise confusion, we may occasionally (inconsistently) call it "feature specification ('Pflichtenheft')", "requirements specification ('Lastenheft')", or "non-functional requirements".

- Consider the following examples:

  - Precise, not realisable: "the list of participants should be sorted conveniently".
  - Precise, testable: "the list of participants should be sorted by immatriculation number, lowest number first".
  - Vague: "how are things to be done?".

- A requirements specification should always be as complete and correct as possible. For analysing an existing requirements/feature specification "finding out what the exact requirements are" is in almost all cases not a requirement but a service provided by a consultant. It is not a solution but should serve as a basis on which the software development is based. To maximise confusion, we may occasionally (inconsistently) call it "feature specification ('Pflichtenheft')", "requirements specification ('Lastenheft')", or "non-functional requirements". The source of requirements is documented, requirements are uniquely identifiable, and all requirements (existing in somebody's head, or a document, or ... ) should be present.

- In the following (unless otherwise noted), we discuss the analysis of requirements specifications. We have already seen the importance of requirements specifications, and we shall now present analysis techniques.
Kinds of Requirements: Functional and Non-Functional

• Proposal: View software $S$ as a function $S$: $i_1, i_2, i_3, \ldots \mapsto o_0, o_1, o_2, \ldots$ which maps sequences of inputs to sequences of outputs.

Examples:
- Software “compute shipping costs”:
  - $o_0$: initial state,
  - $i_1$: shipping parameters (weight, size, destination, \ldots),
  - $o_1$: shipping costs
  And no more inputs, $S$: $i_1 \mapsto o_1$.
- Software “traffic lights controller”:
  - $o_0$: initial state,
  - $i_1$: pedestrian presses button,
  - $o_1$, $o_2$, \ldots: stop traffic, give green to pedestrians,
  - $i_n$: button pushed again\ldots

Every constraint on things which are observable in the sequences is a functional requirement (because it requires something for the function $S$).
Thus timing, energy consumption, etc. may be subject to functional requirements.

Clearly non-functional requirements: programming language, coding conventions, process model requirements, portability.\ldots

Kinds of Requirements: Hard and Soft Requirements

• Example of a hard requirement:
  - Cashing a cheque over $N$ must result in a new balance decreased by $N$; there is not a micro-cent of tolerance.

• Examples of soft requirements:
  - If a vending machine dispenses the selected item within 1 s, it’s clearly fine; if it takes 5 min., it’s clearly wrong — where’s the boundary?
  - A car entertainment system which produces “noise” (due to limited bus bandwidth or CPU power) in average once per hour is acceptable, once per minute is not acceptable.

The border between hard/soft is difficult to draw, and as developer, we want requirements specifications to be “as hard as possible”, i.e. we want a clear right/wrong.
As customer, we often cannot provide this clarity; we know what is “clearly wrong” and we know what is “clearly right”, but we don’t have a sharp boundary.
→ intervals, rates, etc. can serve as precise specifications of soft requirements.

Kinds of Requirements: Open and Tacit

• open: customer is aware of and able to explicitly communicate the requirement,
• (semi-)tacit: customer not aware of something being a requirement (obvious to the customer but not considered relevant by the customer, not known to be relevant).

Examples:
- buttons and screen of a mobile phoneshould be on the same side,
- important web-shop items should be on the right hand side because the main users are socialised with right-to-left reading direction,
- the ECU (embedded control unit) may only be allowed use a certain amount of bus capacity.

Analyst knows domain new to domain
Customer/Client explicit requirements discovered requirements discoverable requirements
(semi-)tacit requirements discoverable requirements discoverable with difficulties tacit hard/impossible to discover (Gacitua et al., 2009)
• distinguish don’t care: intentionally left open to be decided by developer.
### Requirements Analysis Techniques

(A Selection of) Analysis Techniques

- Focus
  - current desired innovation

- Analysis of existing data and documents
  - Observation
  - Questioning with (closed structured, open questions)
  - Interview
  - Modelling
  - Experiments
  - Prototyping
  - Participative development

(Ludewig and Lichter, 2013)

---

### Requirements Elicitation

- Observation:
  - Customers can not be assumed to be trained in stating/communicating requirements.

- It is the task of the analyst to:
  - ask what is wanted,
  - ask what is not wanted,
  - establish precision,
  - look out for contradictions,
  - anticipate exceptions, difficulties, corner-cases,
  - have technical background to know technical difficulties,
  - communicate (formal) specification to customer,
  - "test" own understanding by asking more questions.

→ i.e. to elicit the requirements.

Goal: automate opening/closing of a main door with a new software.

A made up dialogue.

Analyst: So in the morning, you open the door at the main entrance?
Customer: Yes, as I told you.
A: Every morning?
C: Of course.
A: Also on the weekends?
C: No, on weekends, the entrance stays closed.
A: And during company holidays?
C: Then it also remains closed of course.
A: And if you are ill or on vacation?
C: Then Mr. M opens the door.
A: And if Mr. M is not available, too?
C: Then the first client will knock on the window.
A: Okay. Now what exactly does "morning" mean?

(Ludewig and Lichter, 2013)

---

### How Can Requirements Engineering Look In Practice?

- Set up a core team for analysis (3 to 4 people), include experts from the domain and developers. Analysis benefits from highest skills and strong experience.

- During analysis, talk to decision makers (managers), domain experts, and users. Users can be interviewed by a team of 2 analysts, ca. 90 min.

- The resulting "raw material" is sorted and assessed in half- or full-day workshops in a team of 6-10 people. Search for, e.g., contradictions between customer wishes and for prioritisation.

- Note: The customer decides. Analysts may make proposals (different options to choose from), but the customer chooses. (And the choice is documented.)

- The "raw material" is basis of a preliminary requirements specification (audience: the developers) with open questions. Analysts need to communicate the requirements specification appropriately (explain, give examples, point out particular corner-cases). Customers without strong maths/computer science background are often overstrained when "left alone" with a formal requirements specification.

- Result: dictionary, specified requirements.

- Many customers do not want (radical) change, but improvement.

- Good questions: How are things done today? What should be improved?

---

(Section of required terminologies)

Requirements Documents are important—e.g., for negotiation, design & implementation, documentation, testing, delivery, re-use, re-implementation.

A Requirements Specification should be correct, complete, relevant, consistent, neutral, traceable, objective.

Note: vague vs. abstract.

Requirements Representations should be easily understandable, precise, easily maintainable, easily usable.


It is the task of the analyst to elicit requirements.

Natural language is inherently imprecise, counter-measures:

natural language patterns.

Do not underestimate the value of a good dictionary.

References


