Requirements engineering

Paul Jackson

School of Informatics
University of Edinburgh

Kinds of requirements

Functional requirements (*services*): What the system should do.

Non-functional requirements (*constraints* or *quality reqs.*): How fast it should do it; how seldom it should fail; what standards it should conform to; how easy it is to use; etc.

Non-functional requirements may be more important than functional requirements!

- Can be workarounds for functional requirements
- User experience often shaped by non-functional.

Distinction not always clear-cut

- Security might initially be a non-functional requirement, but, when requirements refined, it might result in addition of authorisation functionality

Requirements vs. design

Requirements try to avoid design,

- expressing what is desired,
- not how what is desired should be realised

Stakeholders in requirements

Requirements are usually relevant to multiple stakeholders:

- End users
- Customers paying for software
- Government regulators
- System architects
- Software developers
- Software testers
- ...

Requirements capture processes

Process activities include

- Gathering (*elicitation*)
- Sorting out (*analysis*)
- Writing down (*specification*)
- Checking (*validation*)

Activities often overlapping, not in strict sequence, and iterated.

Several approaches possible for each activity. Choice is very dependent on nature of software developed and overall software development process.

Requirements capture is critical

- Faulty requirement capture can have huge knock-on consequences in later software process activities.
  - It is the major source of project failure according to Standish CHAOS reports.
- One motivation for incremental nature of Agile processes.

Requirements elicitation sources

- Goals: high-level objectives of software
- Domain Knowledge: Essential for understanding requirements
- Stakeholders: Vital, but they may find expressing requirements difficult
- Business rules: E.g. Uni regulations for course registration
- Operational Environment: E.g. concerning timing and performance
- Organisational Environment: How does software fit with existing practices?

(From SWEBOK V3, Ch1)

Requirements elicitation techniques

Techniques include:

- Interviews
- Scenarios
- Prototypes
- Observation

Requirements elicitation: interviews

Traditional method: ask them what they want, or currently do

Can be challenging:

- Jargon confusing
- Interviewees omit information obvious to them

Important to

- be open minded: requirements may differ from those pre-conceived,
- use leading questions to focus dialogue. e.g. from first-cut proposal for requirements.

Requirements elicitation: scenarios

Scenarios are typical possible interactions with the system

- Provide a context or framework for questions.
- Allow "what if" or "how would you do this" questions.
- Easy for stakeholders to relate to
- Can be captured as user stories and use cases

Requirements elicitation: prototypes

Can include

- screen mock-ups
- storyboards
- early versions of systems

Like scenarios, but more “real”. High quality feedback. Often help to resolve ambiguities.

Requirements elicitation: observation

Suitable if replacing existing system or business process

- Nuances can make or break a software product

Immersive method. Expensive.

Helps with:

- Surfacing complex / subtle tasks and processes
- Finding the nuances that people never tell you

Not so good if innovating

- Consider 15 years ago capturing requirements for a touchscreen smartphone

Requirements analysis

Requirements elicitation often produces a set of requirements that

- contains *conflicts* (e.g., one stakeholder wants one-click access to data, another requires password protection)
- is *too large* for all requirements to be implemented.

Requirements analysis is the process of getting to a single *consistent* set of requirements, *classified* and *prioritised* usefully, that will *actually be implemented*.

Requirements specification

Requirements can be recorded in various ways, perhaps using:

- very informal means e.g. handwritten cards, in agile development
- a document written in careful structured English, e.g.

  `3.1.4.4` `The system shall...` *for an essential feature*
  `3.1.4.5` `The system should...` *for a desirable feature*

- use case models with supporting text
- a formal specification in a mathematically-based language.

Phrasing of requirements specifications

Different phrasing of requirements needed for different stakeholders.

- User requirements:
  - High-level.
  - Targeting end users and buyers
  - Pre-contract
- System requirements:
  - Much more detailed.
  - Targeting developers, but still reviewed by users.
  - Post-contract.

Requirements validation

Checks include:

- Validity checks:
  - do requirements reflect real needs?
  - Are they up to date?
- Consistency checks
- Completeness checks
- Realism checks:
  - can requirements be met using time and money budgets?
- Verifiability:
  - is it possible to test that each requirement is met?
  - Applies to both functional and non-functional requirements. Non functional requirements must be measurable.

    Response time must be under 1 sec,
    not *Performance must be good*

User Stories

Used in Agile (lightweight) development processes, e.g. Extreme Programming (XP), to document requirements

User stories are brief, written by the customer on an index card. E.g.

> *10. User A leaves the office for a short time (vacation etc.) and assigns his access privileges to user B, so B can take care of A's tasks while A is gone.*

Pros and cons of user stories

Pros:

- can really be owned by the customer: so more likely to be correct
- quick to write and change
- easy to arrange and organise on a table or wall
  - E.g. could group by priority or risk

Cons:

- May be incomplete, inconsistent
- Only work in conjunction with good access to the customer
- Not suitable to form the basis of a contract

Reading

Required: *SWEBOK V3*, Chapter 1, Software Requirements.

Suggested: *Sommerville*, Part 1 chapter on Requirements Engineering.