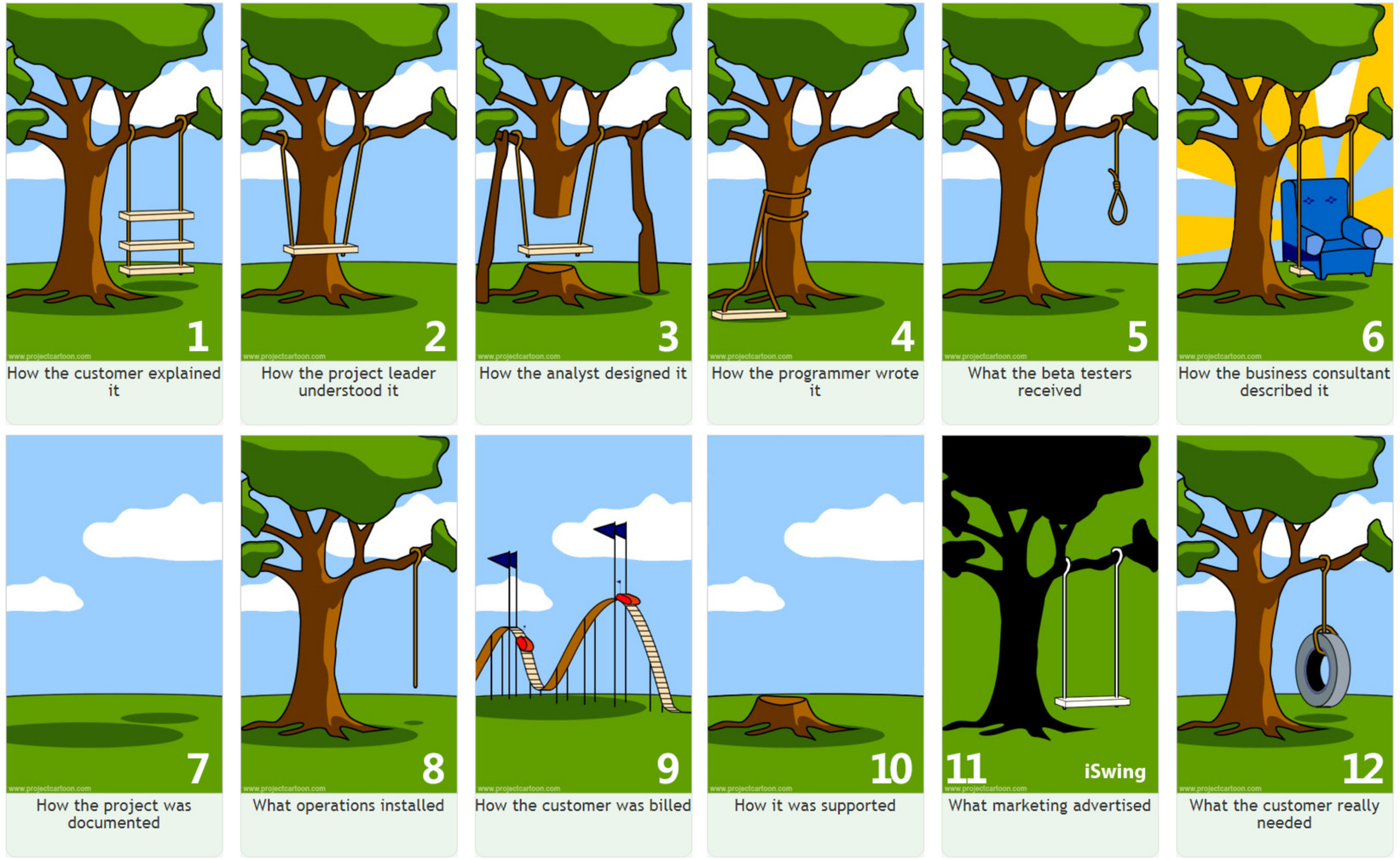
Requirements Specification

(Derived from Prof Damian's originals)



Outline

- Why have a requirements specification
- Quality attributes for requirements and reqts specification

Requirements Specification

- Communication tool between multiple stakeholders
- Communicates understanding of requirements
- Often used as a contract
- Baseline for change control

Documentation vs Specification

- In Cynefin's Complex domain, no need for elaborate "spec". Focus on safe to fail probes/experiments to learn.
- In the Clear/Simple domain, action is obvious. No need to write down requirements in elaborate way (maybe user stories).
- In Complicated domain, many moving pieces, different actors and stakeholders.

Appropriate Specification

- Project A has 1 programmer, 6 months work, Clear domain. 5 page memo.

  » Generate understanding and feedback.

- Project B is 50 programmers, contract acquisition, 2 years work. 500 page SRS.

  » Readers: programmers, management, acquirers, lawyers

  » Agreement on what to build.

How developers see users	How users see developers
Users don't know what they want. Users can't articulate what they want. Users have too many needs that are politically motivated. Users want everything right now. Users can't prioritize needs. Users refuse to take responsibility for the system. Users are unable to provide a usable statement of needs. Users are not committed to system development projects. Users are unwilling to compromise. Users can't remain on schedule.	Developers don't understand operational needs. Developers place too much emphasis on technicalities. Developers try to tell us how to do our jobs. Developers can't translate clearly stated needs into a successful system. Developers say no all the time. Developers are always over budget. Developers are always late. Developers ask users for time and effort, even to the detriment of the users' important primary duties. Developers set unrealistic standards for requirements definition. Developers are unable to respond quickly to legitimately changing needs.

SRS contents

- Functionality
- External interfaces (user, hardware, software, communication)
- Performance
- Design constraints
- Other requirements

SRS should not contain:

- any design or implementation details. These should be described in the design stage of the project.
- the process of producing the software product.

Functional requirements

- inputs => outputs
- user interface structure and behavior
- data processing
- error handling

Non-functional requirements

- physical environment
- users and human factors
- documentation
- resources
- quality assurance
- security
- system interfaces

Quality characteristics of SRS

- Unambiguous
- Complete
- Verifiable
- Consistent
- Modifiable
- Traceable
- Ranked for importance
- Correct

Qualities of specifications

Unambiguous

- Each statement can be read in **exactly** one way

Complete

- Includes all the significant requirements, e.g. related to functionality, performance, design constraints, attributes or external interfaces
- Specifies all the things the system **must (shall)** do
  – and all the things it must not do!
- Conceptual Completeness
  – E.g. responses to **all** classes of input
- Structural Completeness
  – no TBDs!!!

Verifiable

- A process exists to test specification of each requirement

Quality characteristics of requirements specification

Traceable

- The origin of each requirement is clear (“backward traceability”)
- Facilitates referencing to future documentation related to RS (“forward traceability”) such as test plans, design specs

Correct/Valid

- contains as much but no more than what is required (i.e. expresses only the real needs of stakeholders)

Ranked by importance

- Contains priority information for each requirement

Quality characteristics of requirements specification

Consistent

Three types of conflict which can occur are:

- **different terms used for the same object**: e.g. "a P45" and "a tax form" might be used to describe the same form.
- **characteristics of objects conflict**: e.g. in one part of the requirements document, "a red light will indicate a fault", while in another part, "a blue light will indicate a fault".
- **logical or temporal faults**: e.g. "A follows B" in one part, "A and B occur simultaneously" in another.

Modifiable

- Easy to use organization, easy to change without difficulty

Typical Mistakes (Easterbrook)

- **Noise**

  » the presence of text that carries no relevant information to any feature of the problem.

- **Silence**

  » a feature that is not covered by any text.

- **Over-specification**

  » text that describes a feature of the solution, rather than the problem.

More common mistakes

- **Contradiction**

  » text that defines a single feature in a number of incompatible ways.

- **Ambiguity**

  » text that can be interpreted in at least two different ways.

- **Forward reference**

  » text that refers to a feature yet to be defined.

- **Wishful thinking**

  » text that defines a feature that cannot possibly be validated.

More mistakes (3)

- **Jigsaw puzzles**
  - e.g. distributing requirements across a document and then cross-referencing
- **Duckspeak requirements**
  - Requirements that are only there to conform to standards
- **Unnecessary invention of terminology**
  - " E.g., ‘the user input presentation function’, ‘airplane reservation data validation function’

More mistakes (4)

- **Inconsistent terminology**
  - Inventing and then changing terminology
- **Putting the onus on the development staff**
  - i.e. making the reader work hard to decipher the intent
- **Writing for the hostile reader**
  - There are fewer of these than friendly readers