CS1632: Requirements

Wonsun Ahn

What are requirements?

- Specifications of software that define expected behavior
  - Often collected into an SRS, *Software Requirements Specification*
  - The SRS comes in legal binders hundreds of pages long
  - And, yes, the SRS is often legally binding (no pun intended)

- *Requirements engineering*: Managing and documenting requirements

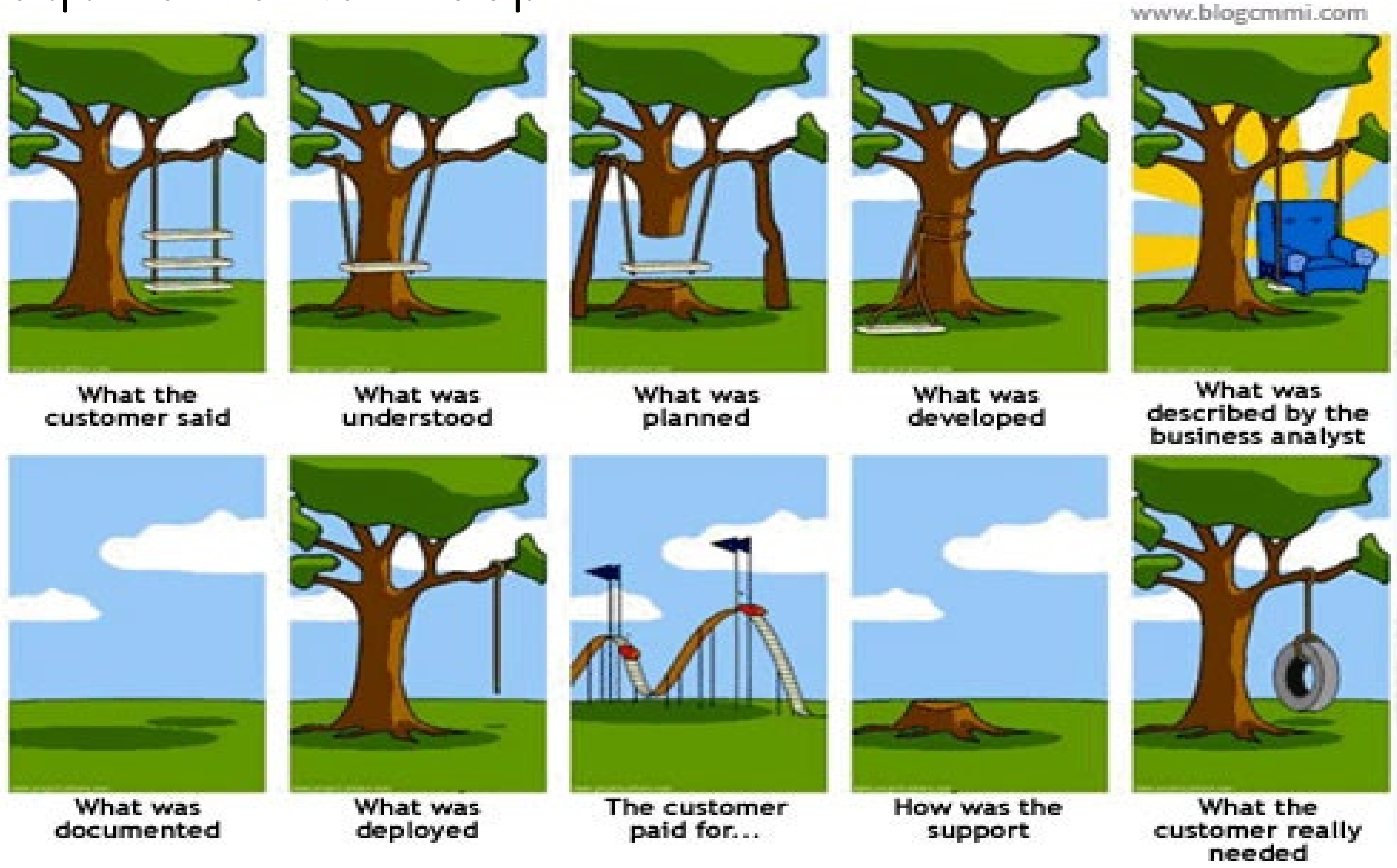
- Requirements drives ⟶ Testing drives ⟶ Development
  - Requirements engineering is where software quality assurance starts

Requirements Evolve

- Requirements are not set in stone and do evolve
  - The testing infrastructure and code implementation must also evolve too
  - Managing requirements is crucial to keep everyone on the same page

- Bad requirements engineering can cause *requirements creep*

Requirements Creep



Requirements Validation

Requirements Validation prevents Requirements Creep

- *Requirements Verification*: Are we building the **software right**? (**testing**)
    1. Derive expected behavior from requirements for each test case
    2. Compare expected behavior with observed behavior

- *Requirements Validation*: Are we building the **right software**?
    1. Interview stakeholders to see if requirements:
        - Match actual stakeholder needs (Validity check)
        - Are technically feasible (Realism check)
        - Requirements are testable (Verifiability check)
    2. Check consistency / ambiguity / completeness

Case Study: Requirements Validation for Bird Cage

- Here are requirements for a bird cage:
  - The cage shall be able to house an ostrich.
  - The cage shall be 2 feet tall.
  - The cage bars shall be made of candy bars.
  - At least 90% of ostriches shall like the cage.

- Are these good requirements?

Validity Check

- Requirements must align with stakeholders needs and wants

- What’s wrong with these requirements?
  - The cage shall be able to house an ostrich.
  - The cage shall be 2 feet tall.
  - The cage bars shall be made of candy bars.
    → Candy bars attract flies. Have the users considered this?
  - At least 90% of ostriches shall like the cage.

Validity Check

- Misconception: Stakeholders know what they want
  Truth: What looks good on paper is often a flop when seen in real life

- Misconception: “More is better” – more features, more customization
  Truth: “Less is more” – usability suffers with “more”

- Misconception: Stakeholders are limited to end-users
  Truth: Stakeholders also include operators, managers, investors, …

☛ Do early prototyping and demos in front of stakeholders

Realism Check

- It must be realistic to **implement** the requirements
  - Must be realistic in terms technology / budget / timeline

- What's wrong with these requirements?
  - The cage shall be able to house an ostrich.
  - The cage shall be 2 feet tall.
  - The cage bars shall be made of candy bars.
    → Cannot make a structurally sound cage with candy bars.
  - At least 90% of ostriches shall like the cage.

Verifiability Check

- It must be feasible to **test** the requirements
  - Must be objectively verifiable within budget and timeline

- What’s wrong with these requirements?
  - The cage shall be able to house an ostrich.
  - The cage shall be 2 feet tall.
  - The cage bars shall be made of candy bars.
  - At least 90% of ostriches shall like the cage.
    → Hard to verify whether an ostrich “likes” the cage (without interviewing it).
    → Better: At least 90% of ostriches shall remain in good health after 1 year.

Consistency Check

- Requirements must be internally consistent
  - Requirements must not contradict each other.

- What’s wrong with these requirements?
  - The cage shall be able to house an ostrich.
  - The cage shall be 2 feet tall. → Inconsistent requirements.
  - The cage bars shall be made of candy bars.
  - At least 90% of ostriches shall like the cage.

Ambiguity Check

- Requirements should not be open to interpretation

- What’s wrong with these requirements?
  - The cage shall be able to house an ostrich. $\rightarrow$ A baby ostrich? Or an adult?
  - The cage shall be 2 feet tall. $\rightarrow$ Is 20 feet okay? Is 2.001 feet okay?
  - The cage bars shall be made of candy bars.
  - At least 90% of ostriches shall like the cage.

Completeness Check

- Requirements should cover all aspects of a system
    - If you care that something should occur a certain way, it should be specified

- What's wrong with these requirements?
    - The cage shall be able to house an ostrich.
    - The cage shall be 2 feet tall.
    - The cage bars shall be made of candy bars.
    - At least 90% of ostriches shall like the cage.
    - How about the shape of the cage? Or the width?
    - How thick should the cage bars be? And how far apart?

Pitfall: Specifying HOW, not WHAT

1. Specifying the “how” may not achieve the “what” user has in mind
   - Users care about what the software does, not how it happens

2. Specifying how restricts developers from improving implementation

3. Specifying how often fails the verifiability test
   - Often source code is not available to end user to verify the “how”
   - Even if available, end users typically lack technical capability to verify

Pitfall: Specifying HOW, not WHAT

- **BAD**: The system shall have dual modular redundancy for all modules.
- **GOOD**: The system shall have a mean-time-to-failure of 100 years.

- **BAD**: The system shall be comprised of 100 CPUs with one CPU dedicated to servicing one user.
- **GOOD**: The system shall support up to 100 concurrent users with a response time of less than 10 ms.

Functional Requirements and Non-functional Requirements

Functional and Non-Functional Requirements

- **Functional Requirements**
  - Specify functional behavior of system (an “output”)
  - The system shall **do** X on input Y.
- **Non-Functional Requirements (Quality Attributes)**
  - Specify overall qualities of system, not a specific behavior
  - The system shall **be** X during operation.
- Note “do” vs “be” distinction!

Quality Attribute Categories

- Reliability
- Usability
- Accessibility
- Performance
- Safety
- Supportability
- Security

You can see why quality attributes are sometimes called "-ility" requirements!

Functional Requirement Examples

- **Req 1:** System shall return "NONE" if no elements match the query.
- **Req 2:** System shall throw an exception on illegal parameters.
- **Req 3:** System shall turn on HIPRESSURE light at 100 PSI.
- **Req 4:** HIPRESSURE light shall be red.
  - Note verb is “be” but it still describes an aspect of functional behavior
  - Same as saying: HIPRESSURE light shall flash red

Quality Attribute Examples

- **Req 1:** The system shall be protected against unauthorized access.
- **Req 2:** The system shall be easily extensible and maintainable.
- **Req 3:** The system shall be portable to other processor architectures.
- **Req 4:** The system shall have 99.999 uptime.
  - Note verb is "have" but it still describes a quality, not an output of the system
  - Same as saying: The system shall be available 99.999 of the time

Quality Attributes are Difficult to Test

- Why? Because they are literally qualitative.
- Can be subjective. (e.g., How reliable is reliable?)
- Often difficult to measure. (e.g., How do you measure reliability?)

Solution

Agree with stakeholders upon quantifiable requirements that ensure quality.

Converting Qualitative to Quantitative

- **Performance:** response time, transactions per second
- **Reliability:** Mean-time-between-failures (MTBF)
- **Robustness**: How many simultaneous failures can system cope with
- **Portability:** Number of systems targeted, or how long it takes to port
- **Usability:** Average amount of time required for training

Qualitative to Quantitative Example

- Quality attributes should be expressed in a quantitative way
  - Or else they are ambiguous
- Example
  - **BAD:** The system shall be highly usable.
  - **GOOD:** Over 90% of users shall be able to operate the software after one hour of training.
- Example
  - **BAD:** The system shall be reliable enough to be used in a space station.
  - **GOOD:** The system shall have a mean-time-between-failures of 100 years.

Now Please Read Textbook Chapters 5

- (Optional) If you are interested in further reading:

  IEEE Recommended Practice for
  Software Requirements Specifications (IEEE Std 830-1998)

- Can be found in resources/IEEE830.pdf in course repository