CS1632: Requirements

Wonsun Ahn

Requirements and Requirements Validation

What are requirements?

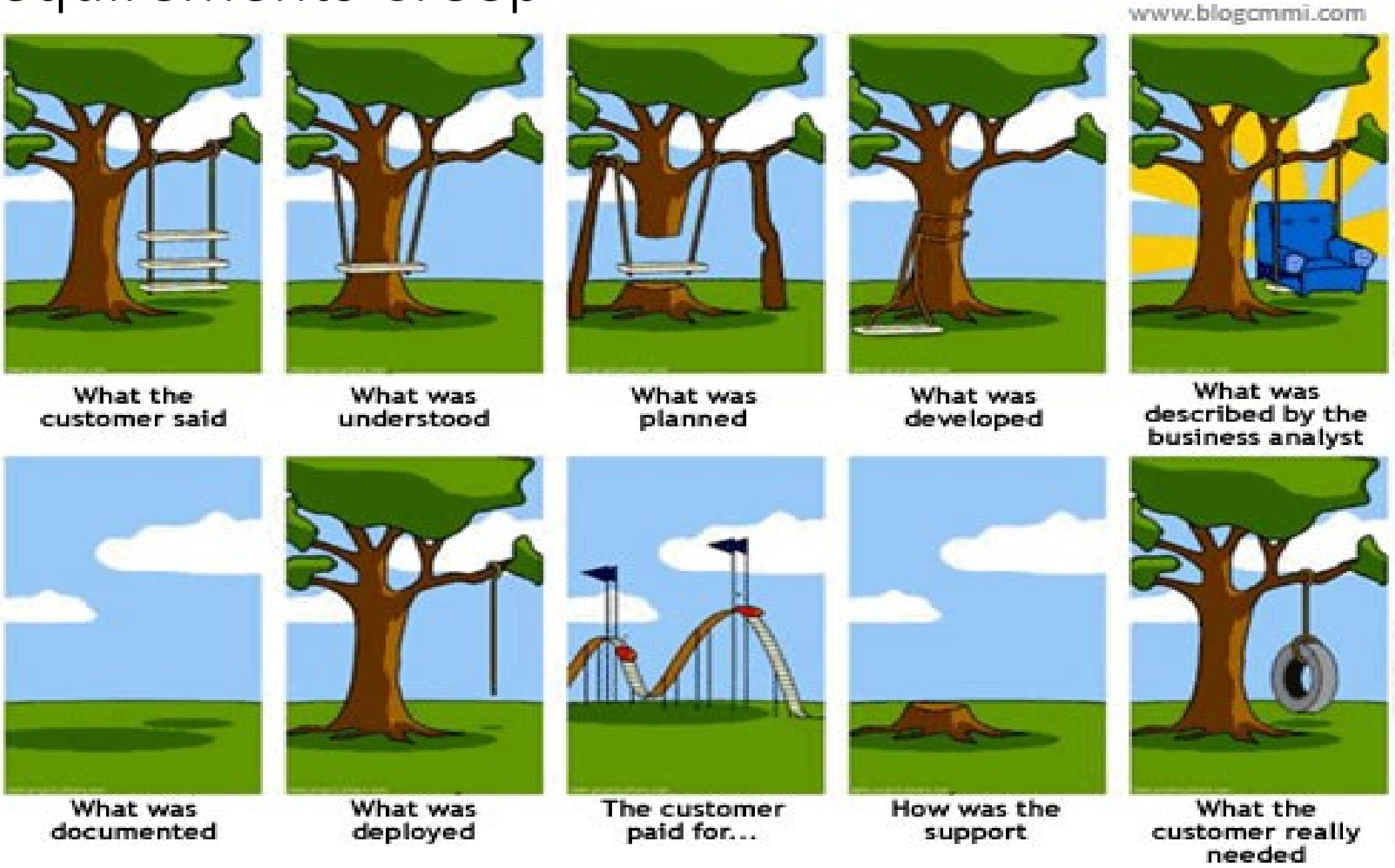
- The specifications of the software
  - Often collected into an SRS, *Software Requirements Specification*
  - The SRS comes in legal binders hundreds of pages long
  - And, yes, the SRS is legally binding (pun intended)
- This is how developers know what code to write
- This is also how testers know what to test

Requirements Evolve

- Requirements are not (usually) set in stone and do evolve
  - Means your code implementation must evolve with the requirements
  - Means your testing infrastructure must also evolve with the requirements
  - A clear understanding of the requirements is crucial at any given point

- *Requirements engineering*: Managing and documenting requirements
  - Also part of QA since low quality requirements result in low quality software
  - Bad requirements engineering can cause *requirements creep*

Requirements Creep



Role of Software QA Engineer

- *Requirements Verification*: Are we building the **software right**?
  (a.k.a. **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. Pore over requirements to make sure they make sense
  2. Interview stakeholders to see if requirements match actual needs
  3. Interview developers to see if requirements are technically feasible
  4. Interview testers to see if requirements are verifiable

Aspects of Requirements Validation

- Does the SRS make sense **internally**?

Completeness check: does SRS cover all aspects of software?

Consistency check: does SRS contain any logical conflicts?

Ambiguity check: does SRS contain room for interpretation?

- Does the SRS make sense **externally**?

Validity check: does SRS align with user needs?

Realism check: is SRS something that can be feasibly implemented?

Verifiability check: is SRS something that can be feasibly tested?

Let’s validate requirements for a Bird Cage

- This what our requirements engineer came up with:
  - 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.

- Will it pass requirements validation?

Completeness Check

- Requirements should cover all aspects of a system
  - Anything not covered is liable to different interpretation
  - 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.

Completeness Check

- Requirements should cover all aspects of a system
  - Anything not covered is liable to different interpretation
  - 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.
  - What about the shape of the cage? Or the width?
  - How thick should the cage bars be? And how far apart?

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.
  - The cage bars shall be made of candy bars.
  - At least 90% of ostriches shall like the cage.

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. → Too small to house an ostrich?
  - 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.
  - 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.

Ambiguity Check

- Requirements should not be open to interpretation

- What's wrong with these requirements?
  - The cage shall be able to house an ostrich. → A baby ostrich? Or an adult?
  - The cage shall be 2 feet tall. → Is 2.001 feet okay? Is 20 feet okay?
  - The cage bars shall be made of candy bars. → What kind? Twix bars? Mars bars?
  - At least 90% of ostriches shall like the cage. → How much do they have to like it?

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.
  - At least 90% of ostriches shall like the cage.

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.
    → Does anyone really want an ostrich in their homes?
  - The cage shall be 2 feet tall.
  - The cage bars shall be made of candy bars.
    → Candy bars look good but they attract flies.
  - At least 90% of ostriches shall like the cage.

Validity Check

- Misconception: “More is better” – more features, more modes, etc.
  Truth: “Less is more” – ease-of-use suffers with “more”

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

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

☛ Do early prototyping and demos in front of stakeholders

Realism Check

- It must be realistic to **implement** the requirements
  - Must be realistic in terms of current technology
  - Must be realistic within the given budget and delivery date

- 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.

Realism Check

- It must be realistic to **implement** the requirements
  - Must be realistic in terms of current technology
  - Must be realistic within the given budget and delivery date

- 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.
    → Is it technically possible to build 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
  - Should be documented to the level where it is testable
  - Should be testable within the given budget and delivery date

- 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.

Verifiability Check

- It must be feasible to **test** the requirements
  - Should be documented to the level where it is testable
  - Should be testable within the given budget and delivery date

- At least 90% of ostriches shall like the cage.
  → Too costly verify whether 90% of all ostriches in the world like it
  → Hard to verify whether ostrich “likes” cage (without interviewing it)

- A better requirement: At least 90 out of 100 randomly sampled ostriches shall remain in good health after living in the cage for 1 year.

Requirements should say WHAT to do, not HOW to do it!

- **GOOD:** The system shall store all logins for future review.
- **BAD:** The system shall use an associative array in a singleton class called AllLoginsForReview to store all logins.

- **GOOD:** The system shall support 100 concurrent users.
- **BAD:** The system shall use a BlockingQueue to store users, with a maximal size of 100.

Requirements should say WHAT to do, not HOW to do it! Why?

- Users care about what the software does, not how it happens (usually)

- Specifying how restricts developers from improving implementation

- Specifying how precludes blackbox testing
  - Means customer cannot verify requirements just by trying out the software
  - Means developer must provide source code to customer in order to verify

Functional Requirements and Non-functional Requirements

Functional and Non-Functional Requirements

- **Functional Requirements**
  - Specify functional behavior of system
  - 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!

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 is still a functional requirement (describes a feature)
  - 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 is still describing an overarching quality
  - Same as saying: The system shall be available 99.999 of the time

Quality Attribute Categories

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

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

Quality Attributes are Difficult to Test

- Why? Because they are qualities.

- Often difficult to measure.
- Can be very subjective.

Solution

Agree with stakeholders upon quantifiable requirements that ensure quality.

Converting Qualitative to Quantitative

- **Performance:** transactions per second, response time
- **Reliability:** Mean time between failures
- **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
- **Accessibility**: Percentage of population who can use system

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 have no questions using 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