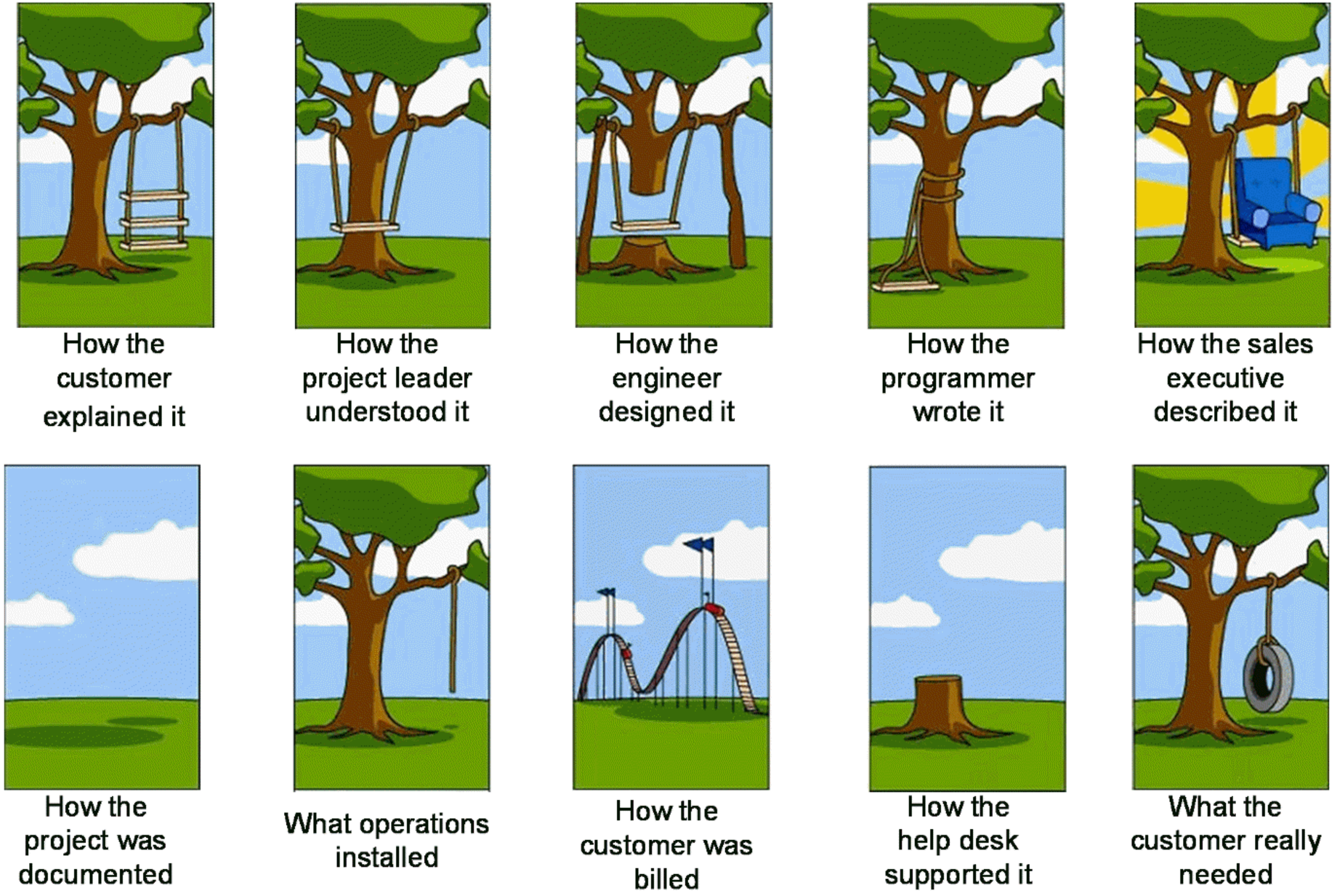
Requirements 1: Overview and Concepts

Claire Le Goues

September 17, 2020



How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the help desk supported it
What the customer really needed

Administrivia

- Calling into Section B
- A new fork of mayan-edms is available to you for HW2!
- HW2 PDF has been updated, and messages on general/announcements should help.
- Should produce a container where changes to the python + main/base.css will reflect immediately in the app.
    - Not included in the mounted volume: included dependencies, bootstrap, or any of the special/page-specific css.
    - But: adding those is straightforward.

Examples adapted arbitrarily from prior years without identifying information!

REFLECTIONS ON REFLECTIONS

Reflection documents

Shallow

- Recite facts about what happened without adding anything.
- Recite statements from class without connecting to experience.
- State lessons learned without any reason why.

Good

- Extrapolate from the facts to add insight.
- Meaningfully connect prior experience or class material to assignment experience.
- Support lessons learned with evidence.

Shallow reflection examples

[PROCESS]

At our first meeting, we developed an initial outline of our approach. This was followed by preparing a list of tasks which were required for implementing the X system. Next, we divided the tasks among ourselves and came up with a rough timeline of the process to be followed."

[SCHEDULE]

"Although we managed to meet all the milestones and implement all the desired features, the exact dates for the same could not be followed towards the end."

[PLANNING]

"Learning how to use API X took a little longer than expected, which caused a setback of a day; but overall we managed to complete the entire project before the deadline and adhered to the timeline."

[TEAM WORK / COMMUNICATION]

"We all agreed to use tool Y to keep in touch. We used it to announce when we started or completed individual tasks, current milestone statuses.. We also used Y to schedule a group meeting for the integration portion of our coding assignment"

Good reflection examples

[PLANNING / PROCESS]

“Since I was interested in the planning, we decided as a team I would be in charge of documenting our progress.. It worked really well to have one person managing what needed to get done or who needed to do it, and ensuring a shared single vision and set of goals as a group. However, there exist negatives approaching things this way…I found that my teammates sometimes would rely on me too heavily.”

[TEAM WORK / COMMUNICATION]

“An example of something that [would] work well is...issue tracking – something I asked them to do since first meeting. It’s easy to forget this information over time... If we had simply reminded ourselves on a regular basis, we would have had fewer problems forgettng these things.”

[PLANNING]

“People seemed to be annoyed because X “was not doing any work”. I believe X did the least amount of work, but we also assigned X the least amount of work. I wonder if this can all be traced back to the fact that X could not attend our first group meeting”

More good examples

[TEAMWORK]

"It helps to say 'thank you' before complaining about a teammate's work. Only take conflict-inducing action if you think it is extremely important and are willing to follow up. Otherwise, you are wasting everyone's time. Would we have treated each other differently if we had known we would be partnered up on more than just this assignment for the class?"

[TEAMWORK]

"two takeaways I had from this project are :

– It is best to present yourself as someone who is willing to help out, and do more than what was originally asked of you. This way, if people decline your offer to help out, they will be okay with the fact that you may not be working as hard as them at that point in time.

– Respect other people's time and work, and take that into consideration when you decide to criticize their work or bring up issues. "

Also

- The homework document includes bulleted lists and prose outlining what a “good solution” looks like.
- Consider checking your submission against it, at the very least before submitting, if not sooner.

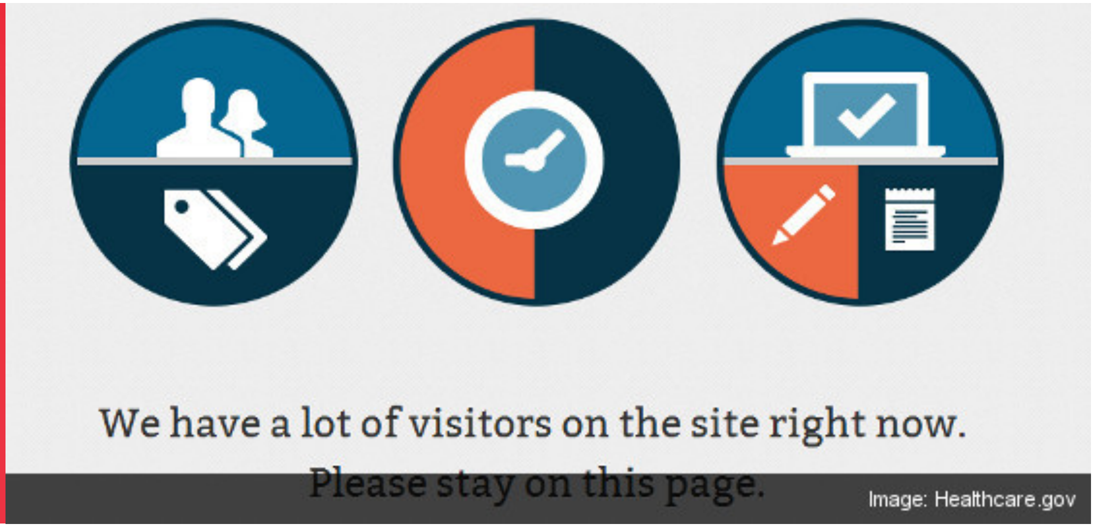
Learning goals

- Explain the importance and challenges of requirements in software engineering.
- Explain how and why requirements articulate the relationship between a desired system and its environment. Identify assumptions.
- Distinguish between and give examples of: functional and quality requirements; informal statements and verifiable requirements.
- State quality requirements in measurable ways

Overly simplified definition.

Requirements say what the system will do (and not how it will do it).

Healthcare.gov



Fred Brooks, on requirements.

- *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.*
- *No other part is as difficult to rectify later.*

— Fred Brooks

A problem that stands the test of time…

A 1994 survey of 8000 projects at 350 companies found: 31% of projects canceled before completed; 9% of projects delivered on time, within budget in large companies, 16% in small companies.

- Similar results reported since.

Causes:

1. Incomplete requirements (13.1%)
2. Lack of user involvement (12.4%)
3. Lack of resources (10.6%)
4. Unrealistic expectations (9.9%)
5. Lack of executive support (9.3%)
6. Changing requirements and specifications (8.7%)
7. Lack of planning (8.1%)
8. System no longer needed (7.5%) .

WHY IS THIS HARD?

Communication problem

Goal: figure out what should be built.

Express those ideas so that the correct thing is built.

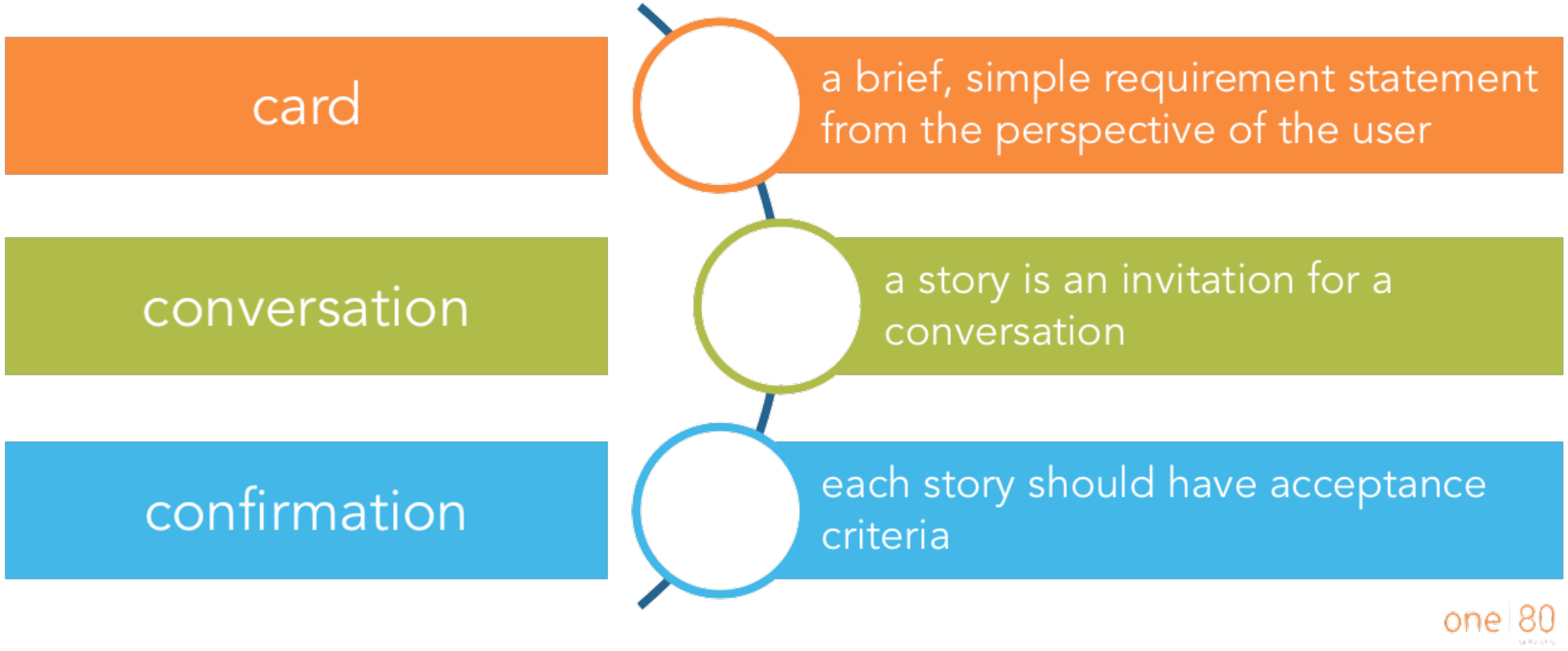
Overall problems

- Involved subproblems?
- Required functionality?
- Nice to have functionality?
- Expected qualities?
- How fast to deliver at what quality for what price?

https://vimeo.com/41800652

EXAMPLE

User Stories



Source: http://one80services.com/user-stories/writing-good-user-stories-hint-its-not-about-writing/

The card

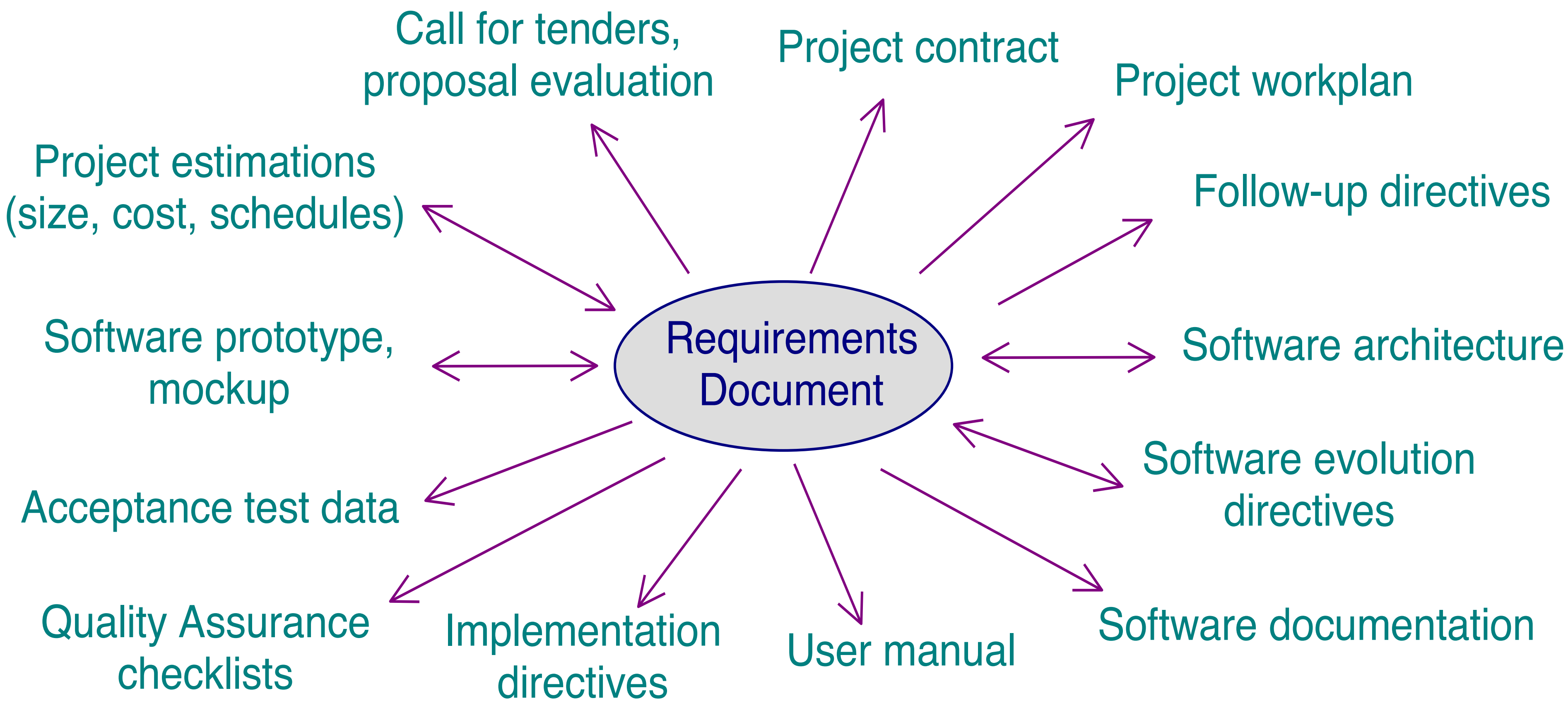
- “As a [role], I want [function], so that [value]”
- Should fit on a 3x5 card

How to evaluate user story?

Source: http://one80services.com/user-stories/writing-good-user-stories-hint-its-not-about-writing/

one 80

Requirements in software projects



Less simplified definition: online shopping

- Stories: Scenarios, Use Cases, and user stories

  "After the customer submits the purchase information and the payment has been received, the order is fulfilled and shipped to the customer's shipping address."

- Optative statements

  *The system **shall** notify clients about their shipping status*

- Domain Properties and Assumptions

  Every product has a unique product code

  Payments will be received after authorization

What is requirements engineering?

- Knowledge **acquisition** – how to capture relevant detail about a system?
  - Is the knowledge complete and consistent?
- Knowledge **representation** – once captured, how do we express it most effectively?
  - Express it for whom?
  - Is it received consistently by different people?
- You may sometimes see a distinction between the requirements *definition* and the requirements *specification*.

Functional Requirements

- What the machine should do
  - Input
  - Output
  - Interface
  - Response to events
- Criteria:
  - Completeness: All requirements are documented
  - Consistency: No conflicts between requirements
  - Precision: No ambiguity in requirements

Quality/Non-functional requirements

- Specify not the functionality of the system, but the quality with which it delivers that functionality.
- Can be more critical than functional requirements
  - Can work around missing functionality
  - Low-quality system may be unusable
- (We'll come back to these in a bit.)

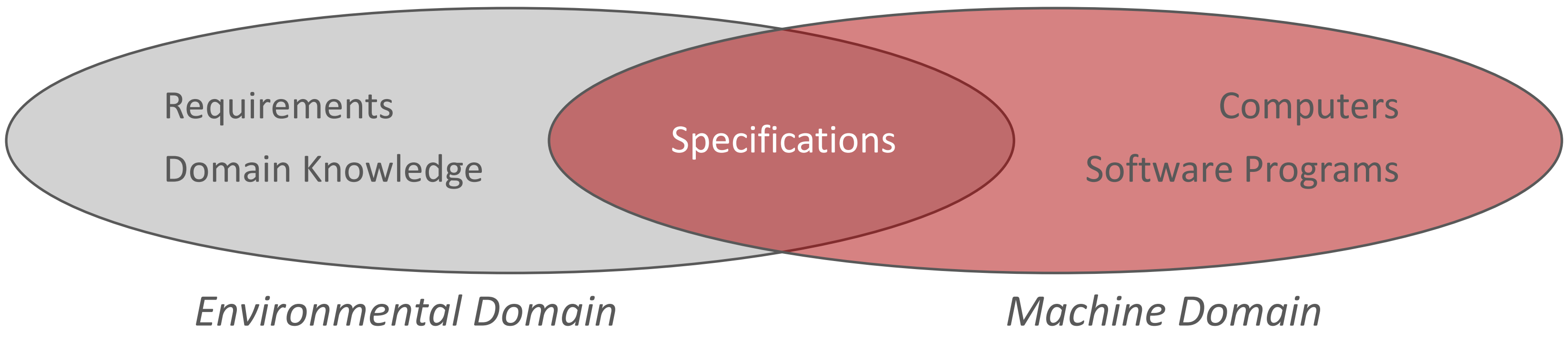
Functional requirements and implementation bias

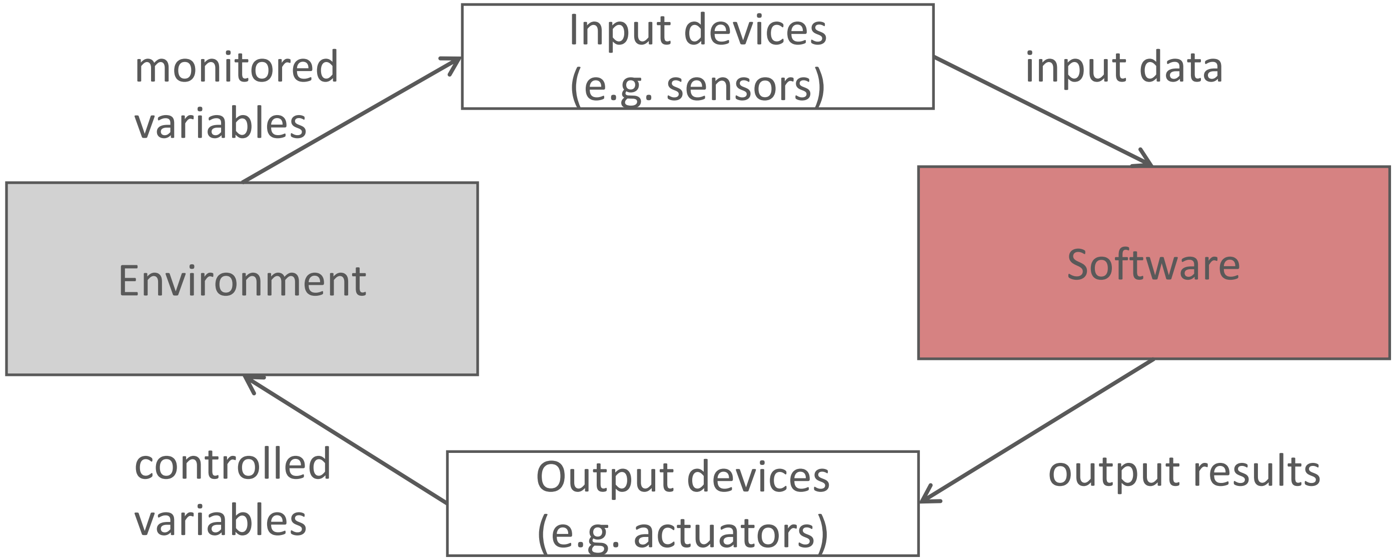
Requirements say what the system will do (**and not how it will do it**).

Why not "how"?

THE WORLD AND THE MACHINE

Environment and the Machine





Pamela Zave & Michael Jackson, "Four Dark Corners of Requirements Engineering," *ACM Transactions on Software Engineering and Methodology,* 6(1): 1-30, 1997.

Actions of an ATM customer:

withdrawal-request(a, m)

Properties of the environment:

balance(b, p)

Actions of an ATM machine:

withdrawal-payout(a, m)

Properties of the machine:

expected-balance(b, p)

What other **models of the world** do machines maintain?

Domain knowledge

- Refinement is the act of translating requirements into specifications (bridging the gap!)
- Requirements: desired behavior (effect on the environment) to be realized by the proposed system.
- Assumptions or domain knowledge: existing behavior that is unchanged by the proposed system.
  - Conditions under which the system is guaranteed to operate correctly.
  - How the environment will behave in response to the system's outputs.

Some gaps must remain…

- Unshared actions cannot be accurately expressed in the machine
  - People can jump over gates (enter without unlocking)
  - People can steal or misplace inventory
- Future requirements are also not directly implementable
  - Phone system: “After all digits have been dialed, do *ring-back, busy-tone* or *error-tone*.”
  - …how do you know the user is done dialing?

Assumptions?

IMPLEMENTATION BIAS

Requirements say what the system will do (**and not how it will do it**).

Why not “how”?

Avoiding implementation bias

- Requirements describe what is observable at the environment-machine interface.
- Indicative mood describes the environment (as-is)
- Optative mood to describe the environment with the machine (to-be).

This can be subtle…

- "The dictionary shall be stored in a hash table" vs. "the software shall respond to requests within 5 seconds."
- Instead of "what" vs. "how", ask "is this requirement only a property of the machine domain?"
- Or is there some application domain phenomenon that justifies it?

QUALITY REQUIREMENTS

Functional Requirements

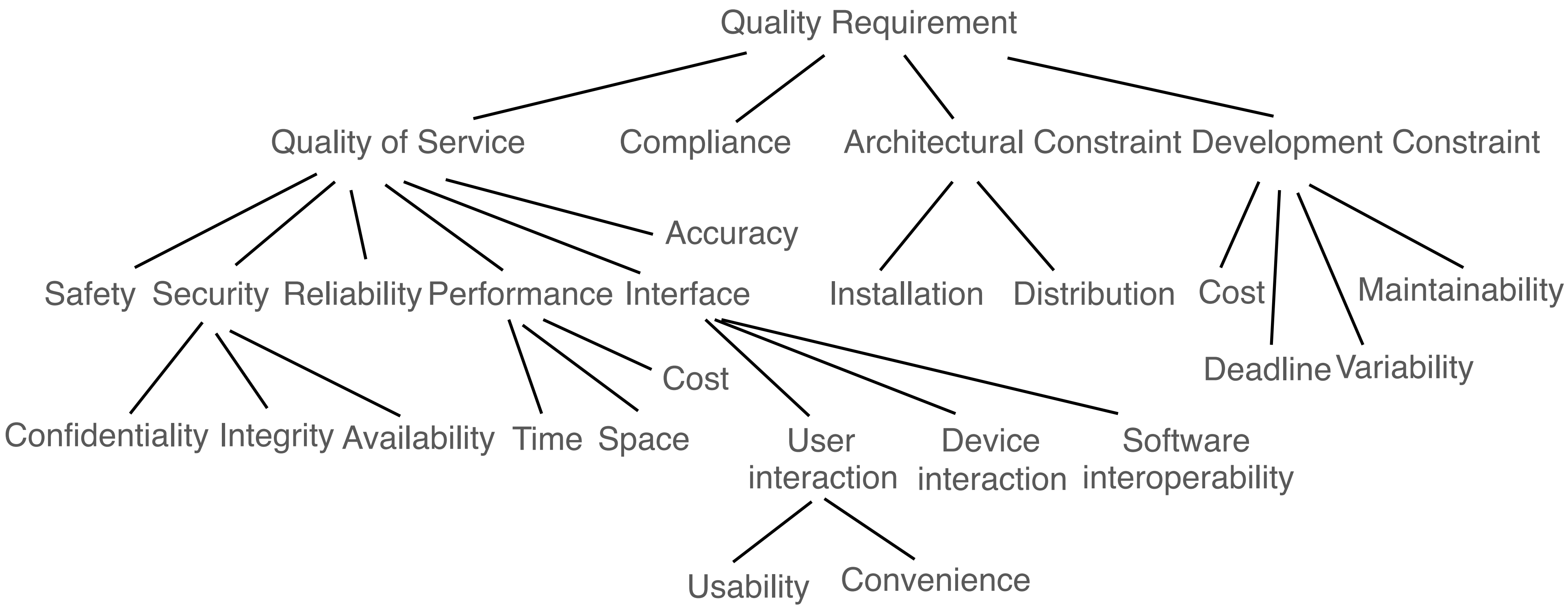
- What the machine should do
  - Input
  - Output
  - Interface
  - Response to events
- Criteria
  - Completeness: All requirements are documented
  - Consistency: No conflicts between requirements
  - Precision: No ambiguity in requirements

Quality (non-funct.) requirements

- Specify not the functionality of the system, but the quality with which it delivers that functionality.
- Can be more critical than functional requirements
  - Can work around missing functionality
  - Low-quality system may be unusable
- Examples?

Here's the thing…

- Who is going to ask for a slow, inefficient, unmaintainable system?
- A better way to think about quality requirements is as *design criteria to help choose between alternative implementations.*
- Question becomes: to what extent must a product satisfy these requirements to be acceptable?



Selling videos on the web?

Expressing quality requirements

- Requirements serve as contracts: they should be testable/falsifiable.
- Informal goal: a general intention, such as ease of use.
  - May still be helpful to developers as they convey the intentions of the system users.
- Verifiable non-functional requirement: A statement using some measure that can be objectively tested.

Examples

- Confidentiality requirement: A non-staff patron may never know which books have been borrowed by others.
- Privacy requirement: The diary constraints of a participant may never be disclosed to other invited participants without his or her consent.
- Integrity req: The return of book copies shall be encoded correctly and by library staff only.
- Availability req: A blacklist of bad patrons shall be made available at any time to library staff. Information about train positions shall be available at any time to the vital station computer.

Source: van Lamsweerde, Requirements Engineering: ... Wiley 2009

Examples

- **Informal goal:** "the system should be easy to use by experienced controllers, and should be organized such that user errors are minimized."
- **Verifiable non-functional requirement:** "Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day, on average."

Exercise: back to simple

- Let’s write some quality requirements!
- Try to write an informal goal, and then turn it into a verifiable non-functional requirement.

Requirements metrics

Property	Measure

ACTIVITIES OF REQUIREMENTS ENGINEERING

Typical Steps

- Identify stakeholders
- Understand the domain
  - Analyze artifacts, interact with stakeholders
- Discover the real needs
  - Interview stakeholders
- Explore alternatives to address needs

Question

- Who is the system for?
- Stakeholders:
  - End users
  - System administrators
  - Engineers maintaining the system
  - Business managers
  - …who else?

Further Reading

- Van Lamsweerde A. Requirements engineering: From system goals to UML models to software. John Wiley & Sons; 2009. Chapter 1