CPSC 310 – Software Engineering

Requirements Gathering and Analysis

Overview

- Introduction
- Requirement types
- Requirements activities
  - Elicitation
  - Analysis
  - Specification
  - Validation

Learning Goals

By the end of this unit, you will be able to:

- Explain why it's important to elicit and specify requirements correctly
- Specify or critique a set of requirements for a project
- Explain advantages and disadvantages of using specific requirement elicitation techniques
- Given a project description, recommend elicitation techniques and stakeholders involved
- Given a particular system, create a comprehensive use case diagram and use cases or comprehensive user stories
- Describe challenges when eliciting and specifying requirements
- Explain why the way we elicit and specify requirements depends on the software process

What are requirements?

- Requirements define:
  - ***What*** the system must do
  - But ***not how*** it should do it (this is design)
- In practice, it is difficult to draw a clear line between ***what*** and ***how***

Why do we need requirements?

- **Understand** what is required of the software
- **Communicate** this understanding precisely to all development parties
- **Control** production to ensure that system meets specs

Why do we need requirements?

Business needs

- Cost estimation
- Budgeting
- Scheduling project

Technical needs

- Software design
- Software testing

Communication needs

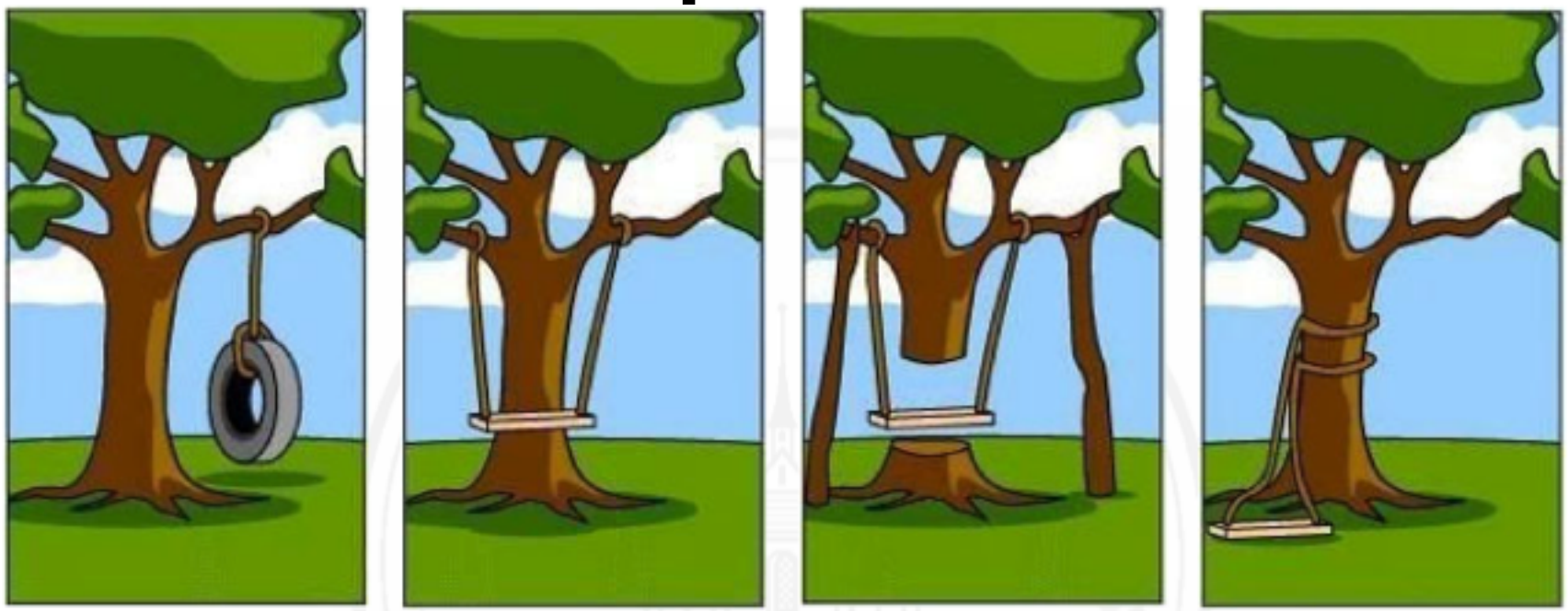
- Documentation and training manuals

Requirements are important

"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***, including all the interfaces to people, to machines, and to other software systems. ***No other part of the work so cripples the resulting system if done wrong.*** No other part of the system is more difficult to rectify later."

Brooks, F. *No Silver Bullet: Essence and Accidents of Software Engineering*. IEEE Computer, 20 (4), April 1987. pp. 10–19.

The telephone effect

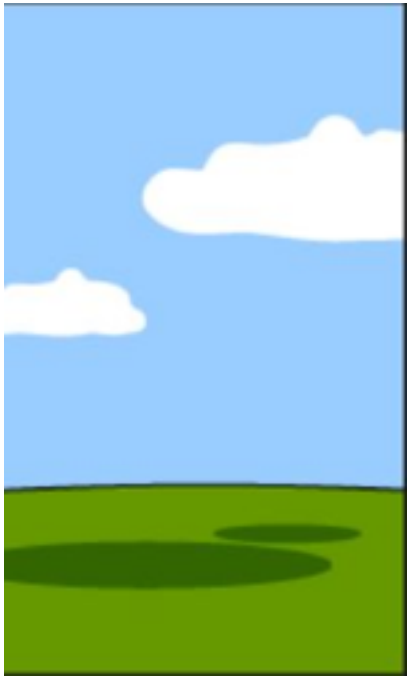


Need	Analysis	Design	Deployed System
What the customer wanted	What the analyst understood	What the architect designed	What the programmers implemented

Even more…

What the customer was billed

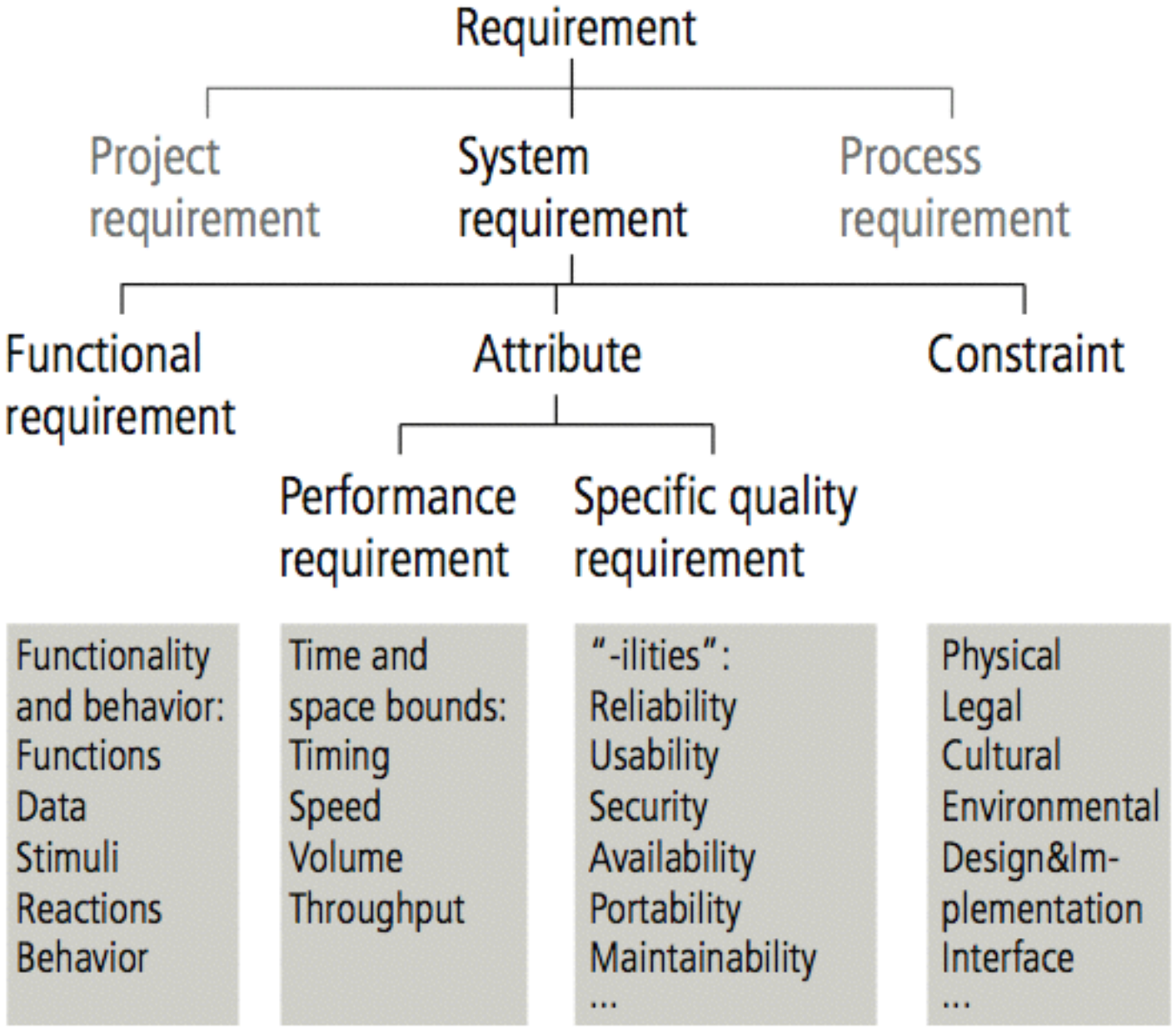
How the business consultant described it



How the project was documented

Classifying requirements

- Old approach: “functional” vs. “non-functional”
- Better approach: constraints, attributes and functions.



Martin Glinz, **On Non-Functional Requirements**, Intl Conf. on Req. Engineering, Delhi, 2007

Discussion question

"I need the car to accelerate to 50 miles per hour in 10s or less"

What type of requirement is this?

A. A functional requirement.

B. A legal constraint.

C. A performance requirement.

D. A quality requirement.

E. A and C.

Requirements Activities

- Elicitation
- Analysis
- Specification
- Validation (for sturdy processes)

Elicitation

We can gather information from existing system documentation, specifications for similar systems and stakeholders.

Elicitation from stakeholders

We need to consider all the different types of stakeholders as they will all have different knowledge of the system.

Stakeholder exercise

List all of the stakeholders that you can for a bank ATM.

example from Sommerville's Software Engineering 8

Elicitation is challenging

- Stakeholders have different needs, which may be in opposition,
  - for example
    - User wants a usable system
    - Customer wants a low-cost system
    - Regulators want a system that is easy to audit
- Stakeholders may not know exactly what they want/need or may find it hard to articulate
- Stakeholders have implicit knowledge of the domain; you have to understand the requirements they describe, possibly with domain-specific terms

Elicitation Techniques

- Questionnaire
- Interviews
- Brainstorm
- Focus groups
- Mock-ups & prototyping
- Ethnographic analysis
- Document study
- …

When a system already exists…

- Document study
  - Read the current documentation
- Ethnographic analysis
- Questionnaires
- Interviews

Interviews

Key point is to pick the ***right people***

- Remember: there are many different stakeholders

Users are expert in their domain, not in SE

- May not know how their needs translate in software

Interview ***in person***

- Not a survey

Context-free questions

- Can be used regardless of the project
  - About the nature of the project
  - About the environment
  - To understand the user profile
- Examples:
  - How is success measured?
  - Who is the user?
  - What problems do you encounter?

Open-ended questions

- Encourages a full, meaningful answer
- Uses the subject's own knowledge
- Good to get a general idea

Closed questions

- Have a short answer
  - Yes/No, or
  - A small number of words
- Good to get a more specific idea or confirmation

Interviews: Have a plan!

- Future questions often depend on previous answer(s)
- But you should have a *template*
  - List of context-free questions
  - A few high-level open-ended questions
  - A clear idea of what you want to know
- Ask the ***general questions first***, then the ***specific questions later***
- Make sure you ask ***clear*** questions (why is this important?)

Interview template

- Establish customer and user profile
  - Name, responsibility, individual measure of success, elements that go against success
- Assess the problem
  - Identify problems without good solutions, cause of problem, current solution, desired solution
- Understand the user environment
  - User background, education, computer literacy
- Recap for understanding
  - Repeat the problem in your own words, ask for feedback, clarification, additions

Class exercise: Example of an interview

- Mom calls up complaining about having too many recipe cards, can't find recipe, can't plan shopping…
- She paid for university, so… you agree to make a recipe system

Class exercise

- Who would you interview?
- What questions would you ask?

(Remember to start general and then get more specific)

Now, trade with the group next to you and see if you can make any suggestions about important questions that they are missing.

Interviews – pros and cons

Advantages:

- Possible to ask clarification or follow up questions
- Rich collection of information (opinions, feelings, goals, hard facts, …)

Drawbacks:

- Interviewing is a difficult skill to master
- Can be time-consuming
- Difficult for people to self-report
  - Mis-remember details
  - Forget implicit details
- Misunderstandings due to lack of domain knowledge

Ethnographic Analysis

- The analyst immerses himself/herself in the work environment and **observes**
- Work is often richer/more complex than suggested (why?)
- Discovery by observation and analysis
  - Workers are *not* asked to explain what they do (why not?)

Ethnographic analysis, Example

When designing a new air control traffic system, observation of the current system led to:

- Controllers often put aircrafts onto potentially conflicting flight paths with the intention to correct them later.
- Existing system raises an audible warning when conflict possible.
- Controllers turned the buzzer off, because they were annoyed by the constant "spurious" warnings.

Sommerville et al.

Ethnographic analysis – pros and cons

Advantages:

·See how people actually work (can be different to what they self-report or task manual describes it)

Drawbacks:

·Can be time-consuming

·People might work differently when being watched

·May miss events that only occur rarely

·Difficult to understand everything that people do from just watching them

Questionnaires

- Good for ***large groups***
- Good using a ***specific*** and ***fixed*** list of questions
- ***Not good*** as the only elicitation technique
  - One-way communication tool
  - Suffer from time-lag (cannot adjust to answers)
  - Selection bias
- Example:
  - Survey of currently used features
  - Reasons for not using specific features

When is elicitation completed?

- When all requirements are elicited?
- When a large portion of them are elicited?

- The "Undiscovered Ruin" problem
  - Try asking an archaeologist: "How many undiscovered ruins are there?"

- Scope the problem to solve, find some ruins, have the stakeholder buy into the requirements

Analysis

- Analyze the results of elicitation
  - Are the answers consistent?
  - Identify trouble spots
  - Identify boundaries
  - Identify most important requirements
- Possibly iterate over elicitation again
- Could need to have stakeholders negotiate

Analysis Example

What is the most important feature to you?

Non-conflicting: OK

Contradiction indicates an issue

- How should you resolve a conflict?

Specification

When using sturdy processes:

- Standard SRS
- Use cases
- Use case diagrams

When using Scrum:

- Product Backlog
- User stories

IEEE Standard 830

Introduction

- Purpose, Scope, Glossary, References, Overview

Overall Description

- Product perspective, Product functions, User characteristics, General constraints, Assumptions and dependencies

Specific Requirements

- System Environment, Functional Requirements, Use Cases, Non-functional Requirements

(These must be understandable by all stakeholders)

Use Case

- A description of the ***possible sequences*** of interactions between ***a system*** and ***its external actors***, related to a ***particular goal***.
- Many use cases for an entire system
- Does not constitute the entire specification
  - Usually included as part of an SRS

Use Case

- Brief use case
  - A few sentences summarizing the use case
- Casual use case
  - One or two paragraphs of text, informal
- Fully-dressed use case
  - A formal document on a detailed template
  - The most common meaning

Casual Use Case

Use Case 1: Buy something

The Requestor initiates a request and sends it to her or his Approver. The Approver checks that there is money in the budget, check the price of the goods, completes the request for submission, and sends it to the Buyer. The Buyer checks the contents of storage, finding best vendor for goods. Authorizer: validate approver's signature . Buyer: complete request for ordering, initiate PO with Vendor. Vendor: deliver goods to Receiving, get receipt for delivery (out of scope of system under design). Receiver: register delivery, send goods to Requestor. Requestor: mark request delivered..

At any time prior to receiving goods, Requestor can change or cancel the request. Canceling it removes it from any active processing. (delete from system?) Reducing the price leaves it intact in process. Raising the price sends it back to Approver.

Cockburn, A. *Writing Effective Use Cases*. Addison-Wesley

Fully-dressed Use Case (1)

Use Case 1: Buy something

Context of use: Requestor buys something through the system, gets it.

Scope: Corporate - The overall purchasing mechanism, electronic and non-electronic, as seen by the people in the company.

Level: Summary

Preconditions: none

Success End Condition: Requestor has goods, correct budget ready to be debited.

Failed End Protection: Either order not sent or goods not being billed for.

Primary Actor: Requestor

Trigger: Requestor decides to buy something.

Main Success Scenario

1. **Requestor**: initiate a request
2. **Approver**: check money in the budget, check price of goods, *complete request for submission*
3. **Buyer**: check contents of storage, find best vendor for goods
4. **Authorizer**: *validate approver's signature*
5. **Buyer**: *complete request for ordering, initiate PO with Vendor*

6. Vendor: deliver goods to Receiving, get receipt for delivery (out of scope of system under design)

7. Receiver: *register delivery*, send goods to Requestor

8. Requestor: *mark request delivered*.

Extensions

1a. Requestor does not know vendor or price: leave those parts blank and continue.

1b. At any time prior to receiving goods, Requestor can change or cancel the request.

Canceling it removes it from any active processing. (delete from system?)

Reducing price leaves it intact in process.

Raising price sends it back to Approver.

2a. Approver does not know vendor or price: leave blank and let Buyer fill in or call back.

2b. Approver is not Requestor's manager: still ok, as long as approver signs

2c. Approver declines: send back to Requestor for change or deletion

3a. Buyer finds goods in storage: send those up, reduce request by that amount and carry on.

3b. Buyer fills in Vendor and price, which were missing: gets resent to Approver.

4a. Authorizer declines Approver: send back to Requestor and remove from active processing.

5a. Request involves multiple Vendors: Buyer generates multiple POs.

5b. Buyer merges multiple requests: same process, but mark PO with the requests being merged.

6a. Vendor does not deliver on time: System does *alert of non-delivery*

7a. Partial delivery: Receiver marks partial delivery on PO and continues

7b. Partial delivery of multiple-request PO: Receiver assigns quantities to requests and continues.

8a. Goods are incorrect or improper quality: Requestor does *refuse delivered goods*. (what does this mean?)

8b. Requestor has quit the company: Buyer checks with Requestor's manager, either *reassign Requestor*, or return goods and *cancel request*.

Fully-dressed Use Case (3)

Deferred Variations
none

Project Information

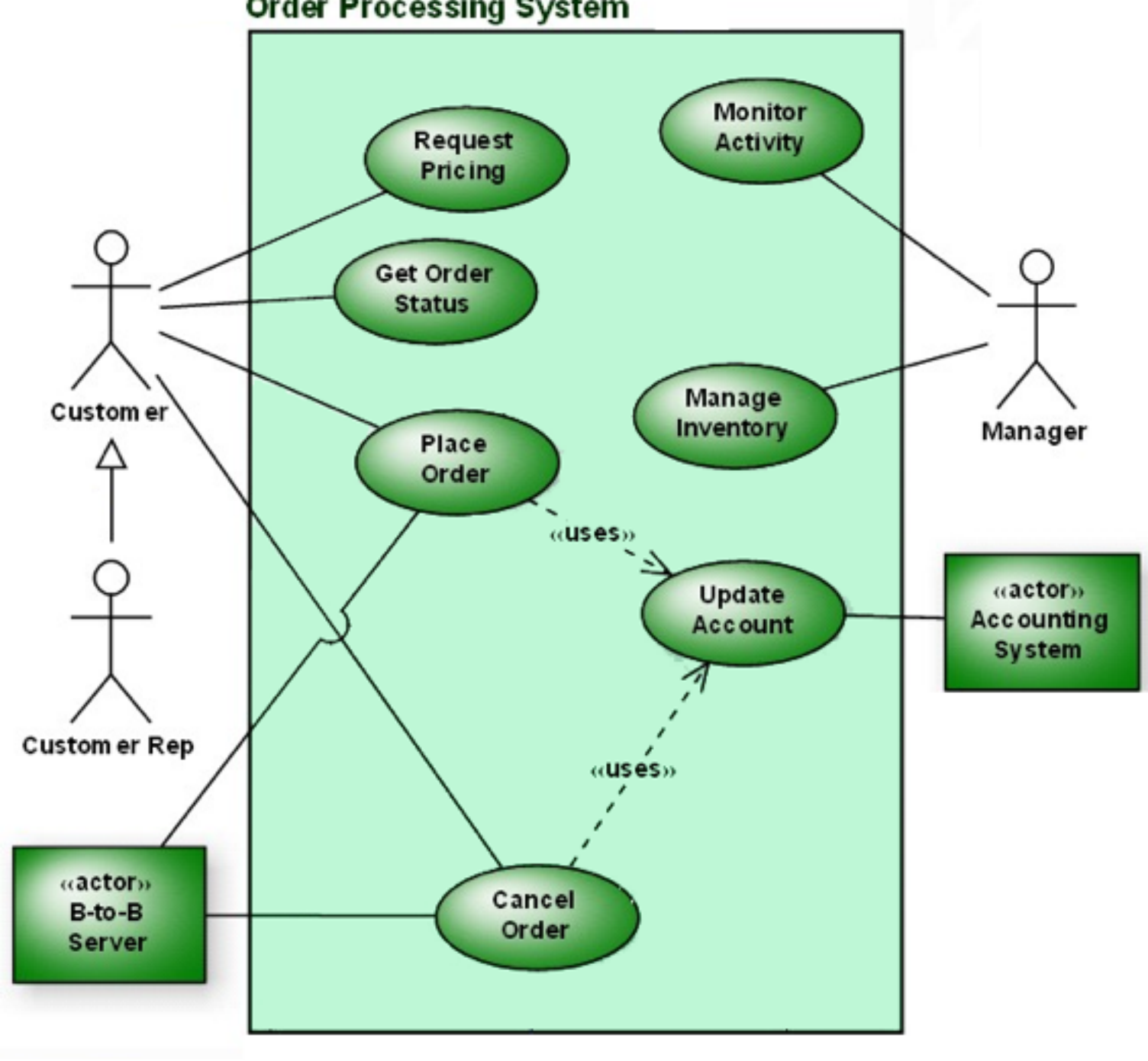
Priority	Release Due	Response time	Freq of use	
Various	Several		Various	3/day

Calling Use Case: none
Subordinate Use Cases: see text
Channel to primary actor: Internet browser, mail system, or equivalent
Secondary Actors: Vendor
Channels to Secondary Actors: fax, phone, car
Open issues

When is a canceled request deleted from the system?
What authorization is needed to cancel a request?
Who can alter a request's contents?
What change history must be maintained on requests?
What happens when Requestor refuses delivered goods?

Use case diagrams

- Show packaging and decomposition of use cases
  - ***Not*** their content
- Each ellipse is a use case
  - Only ***top-level services*** should be shown
  - Not the ***internal behavior***
- Actors can be other systems
- The system (black outline) can be an actor in other use case diagrams
- are ***not enough*** by themselves
  - Must individually document use cases


Order Processing System
Request Pricing
Monitor Activity
Get Order Status
Manage Inventory
Place Order
«uses»
Update Account
«actor» Accounting System
«uses»
Cancel Order
«actor» B-to-B Server
Customer
Customer Rep
Manager

User stories in Agile Dev.

- a ~3 sentence description of what a software feature should do
- written in the customer's language
  - often on an index card
- should only provide enough detail to make a low-risk time estimate (a few days)
- Typically take Role-Goal-Benefit form:
  - "As a <ROLE>, I want to <GOAL> in order to <BENEFIT>"
  - As a student, I need to login to Piazza in order to finish the assignment

User Stories

Examples of User Stories:

- As a user, I want to search for contacts so I can message them.
- As a customer, I want to search for product items so I can buy them.
- As an employer, I want to post a job on the website so people can apply for it.

NOT User Stories:

- Implement contact list view ContactListView.java
- Define the product table database schema
- Automate the job posting algorithm

Product Backlog Item also

- specifies **acceptance or completion criteria** that defines what is meant for this feature to be DONE

- provides **estimate** of the required effort (story points)

- INVEST (for a good user story):
- **I**ndependent, **N**egotiable, **V**aluable to users or customers, **E**stimable, **S**mall, **T**estable

Example – User Story

As a company, I can use my credit card so I can pay for job postings.

- **Note:** Accept Visa, MasterCard, American Express. Consider Discover.

- **Test:** (on the back of the story index card)
- Test with Visa, MasterCard and American Express (pass)
- Test with Diner's Club (fail)
- Test with good, bad and missing card ID numbers.
- Test with expired cards.
- Test with over $100 and under $100.

Example – User Story & Test

- As a Creator, I want to upload a video from my local machine so that any users can view it.

- Note: …
- Test:
- Click the "Upload" button.
- Specify a video file to upload.
  - Check that .flv, .mov, .mp4, .avi, and .mpg extensions are supported.
  - Check that other filetypes aren't able to be uploaded.
  - Check that files larger than 100MB results in an error.
  - Check that movies longer than 10 mins result in an error.
- Click "Upload Video".
- Check that progress is displayed in real time.

What Makes a Good Story

Independent

Negotiable

Valuable to Users or Purchasers

Estimable

Small

Testable

Independent

Little dependence between stories

Keeps development flexible

Makes estimation easier

Negotiable

- Details are negotiated between developers and users
- Cards become reminders of what has been negotiated (especially tests)

Valuable to Purchasers or Users

Customer:

- Purchaser: person who pays for the software

An administrator can determine how many people used the software in the last week.

- User: person who uses the software

A user can search for people by profile attributes

Makes sure customer can estimate value of a story

Not intended to be valuable only to the developers

- The backend database will be MySQL (no value to user; just restricts your options)

Estimable

A developer should be able to estimate how long a story should take to complete

Helps in planning

Problems

- Lack of domain knowledge → ask customer
- Lack of technical knowledge → brush up on tech.
- Story is too big → break it up

Small

Stories should be "just the right size"

Rule-of-thumb: half a day to several days to implement

Too big:

- Estimate inaccurate + no value delivered until story is complete
- Compound → Break up

Too small:

- Writing down the story may take longer than implementing it!
- Combine (staple)

Testable

The story should have a test that goes with it to demonstrate that the story is implemented

Two types

- Automated

  e.g. => JUnit test

- Manual

  e.g. => An untrained user should be able to complete the steps in less than two minutes

User Stories Exercise

- In groups of 2
- Write 2-4 user stories for Amazon
- Include 3 acceptance criteria for each
- Prioritize the user stories 1-4
- Give an hour estimate for implementation
- Additionally,
  - Add a one or two sentence description of a change to the requirements which may cause the implementation to need modification

Validation (for an SRS)

A good requirement specification must be

- Correct
- Complete
- Unambiguous
- Consistent
- Ranked for importance and stability
- Modifiable
- Traceable

Lauesen, S. *Software Requirements*. Addison-Wesley. 2002.

Inspection

Most common errors:

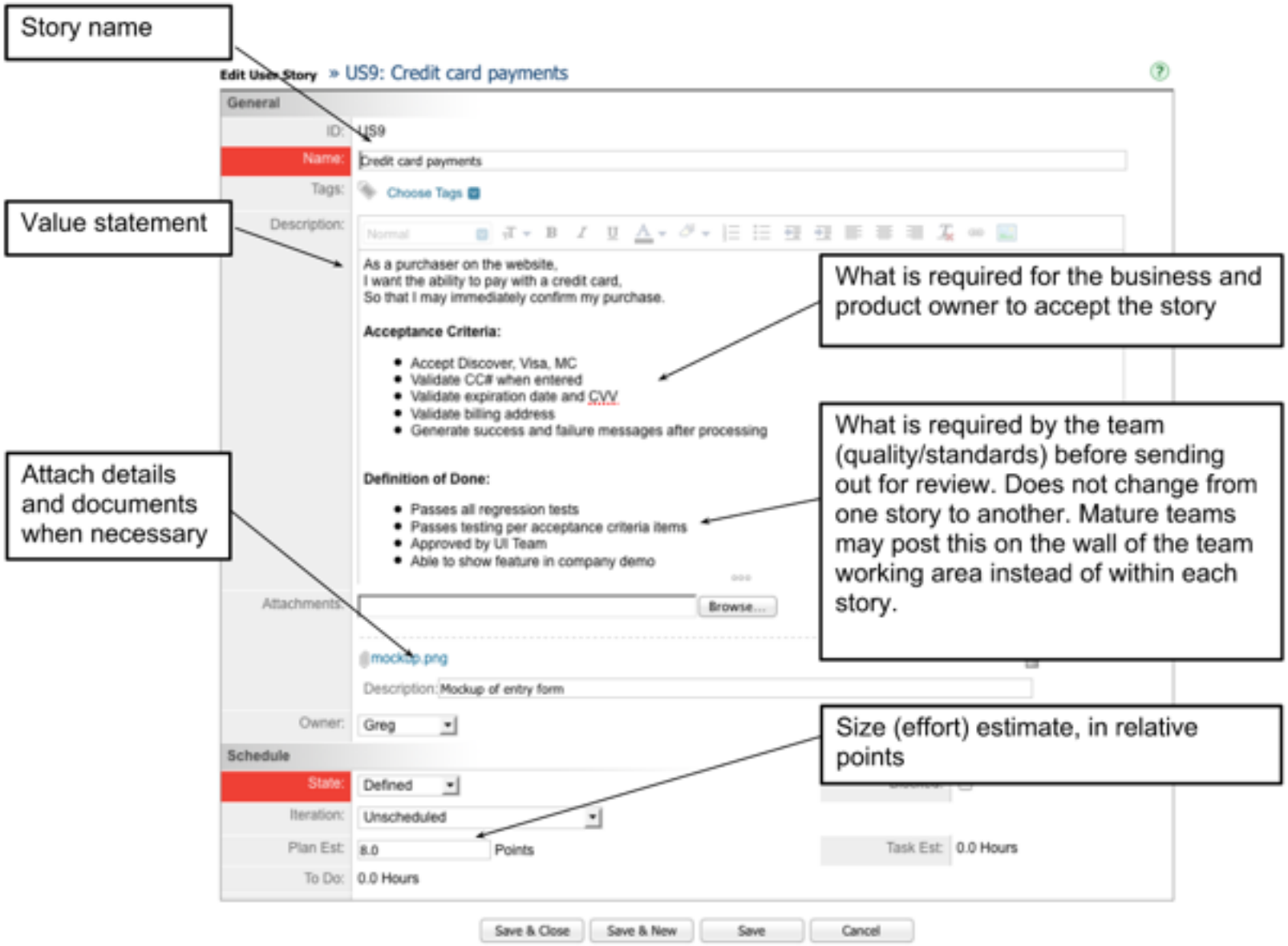
- Omission: A requirement is missing
- Inconsistency: Conflicting requirements
- Incorrect facts
- Ambiguity

The primary tool of validation is inspection

- Should include various stakeholders
  - SRS author, client, designer, end-user, etc.

Validation checklist

1. Do requirements exhibit a clear distinction between function and data?
2. Do requirements define all the information to be displayed to the users?
3. Do requirements address system and user responses to error conditions?
4. Is each requirement stated clearly, concisely and unambiguously?
5. Is each requirement testable?
6. Are there ambiguous or implied requirements?
7. Are there conflicting requirements?
8. Are there area not addressed in the SRS that need to be?
9. Are performance requirements stated?



https://help.rallydev.com/writing-great-user-story