CPSC 310
Requirements

Elicitation, Analysis, Specification and Validation

2014-09-09

Learning goals

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

Explain why it’s important to elicit and specify requirements well

Specify or critique a set of requirements (e.g., user stories) 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 comprehensive user stories

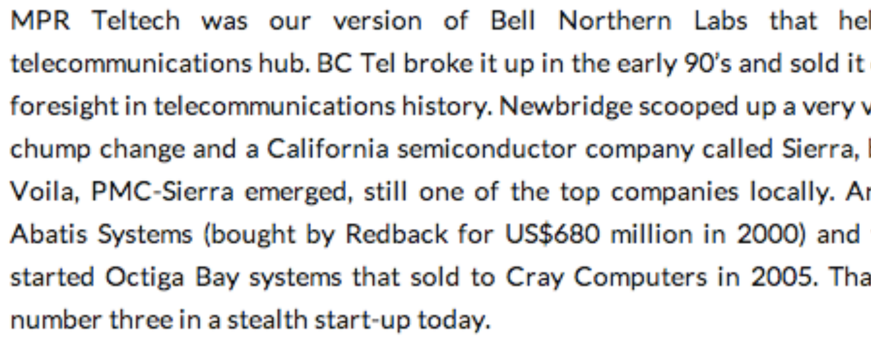
Describe challenges when eliciting and specifying requirements

Who am I and what did you do with Dr. Palyart?

- Professor of Computer Science and Associate Dean (Research & Graduate Studies) in Faculty of Science @ UBC
- co-founder and currently Chief Scientist at Tasktop Technologies Incorporated
- work experience as software developer in telecommunications
- love to swim, skate ski and read and hang out with the family
- Marc had a prior commitment this week so you are stuck with my all of this week

UBC

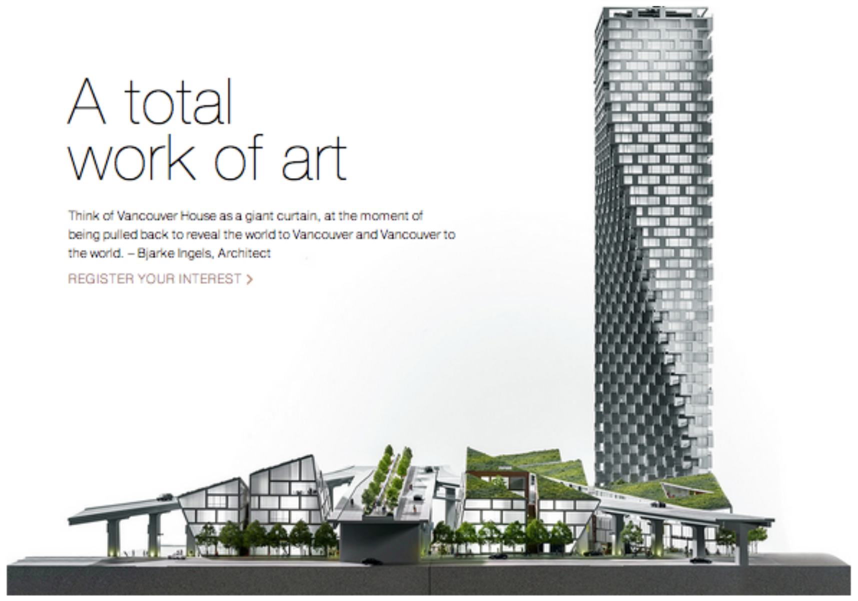
Tasktop



MPR Teltech was our version of Bell Northern Labs that hel
telecommunications hub. BC Tel broke it up in the early 90's and sold it
foresight in telecommunications history. Newbridge scooped up a very v
chump change and a California semiconductor company called Sierra, I
Voila, PMC-Sierra emerged, still one of the top companies locally. Ar
Abatis Systems (bought by Redback for US$680 million in 2000) and
started Octiga Bay systems that sold to Cray Computers in 2005. Tha
number three in a stealth start-up today.

Exercise

- break into groups of 4-6
- imagine that you are working at a software development company that is about to build a software system to operate an elevator in the new Vancouver House development
- in the next 15 min, discuss and write down what the system will need to do
- write it down however you want - I will be walking around and taking some pictures of what you are doing


A total work of art
Think of Vancouver House as a giant curtain, at the moment of being pulled back to reveal the world to Vancouver and Vancouver to the world. – Bjarke Ingels, Architect
REGISTER YOUR INTEREST >

Examples of what you produced...

Requirements

- you were just involved in determining **requirements**
  - about the **“what”** not the **“how”** (which is design)
    - it is not always clear where the “what” ends and the “how” begins!
- requirements are used to
  - understand what is required of the software
  - communicate this understanding to all development parties
  - control production to ensure the system meets the requirements (sometimes requirements are referred to as the specification)

But why do we need requirements?

- Business needs
    - cost estimation
    - budgeting
    - scheduling requests
- Technical needs
    - software design
    - software testing
- Communication needs
    - documentation and training manuals

Requirements activities

- Elicitation
- Analysis
- Specification
- Validation

Elicitation

Elicitation: what is it?

- elicitation is sometimes called requirements gathering
- elicitation is about collecting the requirements from stakeholders
  - who were the stakeholders in the elevator system for Vancouver House?
    - *<you should take notes here… :) >*

Elicitation: stakeholder example

who are the stakeholders if you are going to build the software for an ATM?

<you should take notes again!>

Elicitation: challenges

- what challenges do you see in performing requirements elicitation?
    - *<another place for you to take notes…>*

Elicitation: techniques

- Questionnaires
- Interviews
- Brainstorm
- Focus group
- Mock-ups & prototyping
- Ethnographic analysis
- Documentation study
- …

which techniques are useful
if a similar system already exists
(e.g., elevator system?)

which techniques are useful
if this is the first ever system?

Elicitation: techniques

- **Questionnaires**
- **Interviews**
- Brainstorm
- Focus group
- Mock-ups & prototyping
- **Ethnographic analysis**
- Documentation study
- …

Let's look at a few of the techniques in more depth

Elicitation: questionnaires

- good
  - for large groups
  - when using a specific and fixed list of questions
- not good as the only elicitation technique because
  - one-way communication
  - time-lag (cannot adjust answers)
  - selection bias (only people who feel strongly answer the questionnaire)
- what might be in one?
  - ask whether current features used, prioritize current features, etc.
  - ask what features not used and why

Elicitation: ethnographic analysis

- analyst immerses in work environment and **observes**
  - why not ask the workers to explain what they are doing in the environment?
- why might you find out more than through other approaches?

Elicitation: ethnographic analysis example

- When designing a new air traffic control system, observation of how the air traffic workers worked found:
    - controllers often put aircraft on potentially conflicting flight paths with intention to correct later
    - existing system raised an audible warning when conflict possible
    - controllers turned the buzzer off because they were annoyed by the constant “spurious” warnings

Elicitation: ethnographic analysis: pros and cons

- Pros:
  - *<can you list one pro?>*
- Cons:
  - 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

Elicitation: interviews

- pick the right people who represent a range of stakeholders
- remember users are experts in their domain not in software engineering, it is your job to translate
- interview in person (or high-bandwidth video call)

Elicitation: interviews, kinds of questions

- context-free questions
    - about the project, the environment, the user
    - e.g., how is success measured? who is the user? what problems do you encounter in your work?
- open-ended questions
    - encourage a full and meaningful answer that uses the interviewee's own knowledge
- closed questions
    - have a short answer (e.g., yes/no)
    - good for confirming a specific idea

Elicitation: interviews, need a plan

- have a template
  - list of context-free questions
  - a few high-level open questions
  - a clear idea of what you want to know
- ask the general questions first then the specific questions later *(why is this approach a good idea?)*
- ask clear questions

Elicitation: 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, causes 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, clarifications, additions

Elicitation: interviews exercise

- Mom calls up complaining about having too many recipe cards, can't find recipes, can't plan shopping…
- She's paying your room & board and tuition for University so … you agree to make her an app
- Form a group of 4-6
    - Who would you interview?
    - What questions would you ask?

Elicitation: interviews pros and cons

- Pros:
  - possible to ask clarification or follow-up questions
  - rich collection of information (opinions, feelings, goals, hard facts, etc.)
- Cons:
  - interviewing is a difficult skill to master
  - can be time-consuming
  - difficult for people to self-report
    - mis-remember details
    - forget or don’t realize implicit details
  - misunderstandings due to lack of domain knowledge

Elicitation: when does it end?

- 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

Remember: requirements activities

- Elicitation
- Analysis
- Specification
- Validation

Analysis

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

Remember: Requirements activities

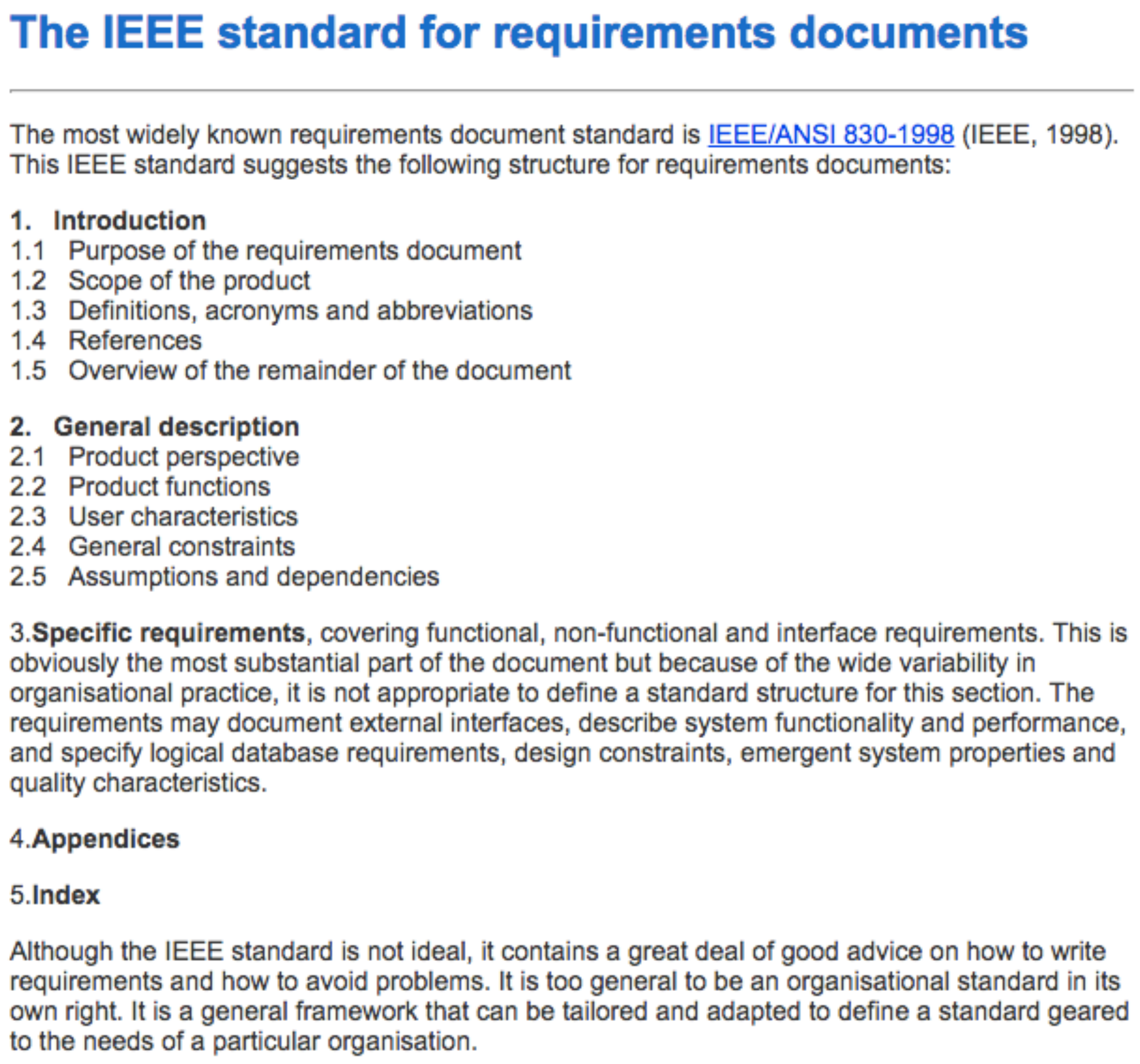
- Elicitation
- Analysis
- Specification
- Validation

Specification

Specification

- There is no one standard or method for specifying (i.e., writing down) requirements
- Different specification methods have different levels of formality
  - the more formal, the more one can precisely state requirements and then verify the implemented system meets the requirements
  - the more formal, the more one might be able to analyze the requirements for consistency, etc.
  - the more formal, typically the more time, not all projects want to spend a lot of time and effort in writing requirements precisely
    - particularly if requirements will change often

Specification: one standard for a requirements document



The IEEE standard for requirements documents

The most widely known requirements document standard is IEEE/ANSI 830-1998 (IEEE, 1998). This IEEE standard suggests the following structure for requirements documents:

1. Introduction
1.1 Purpose of the requirements document
1.2 Scope of the product
1.3 Definitions, acronyms and abbreviations
1.4 References
1.5 Overview of the remainder of the document

2. General description
2.1 Product perspective
2.2 Product functions
2.3 User characteristics
2.4 General constraints
2.5 Assumptions and dependencies

3.**Specific requirements**, covering functional, non-functional and interface requirements. This is obviously the most substantial part of the document but because of the wide variability in organisational practice, it is not appropriate to define a standard structure for this section. The requirements may document external interfaces, describe system functionality and performance, and specify logical database requirements, design constraints, emergent system properties and quality characteristics.

4.**Appendices**

5.**Index**

Although the IEEE standard is not ideal, it contains a great deal of good advice on how to write requirements and how to avoid problems. It is too general to be an organisational standard in its own right. It is a general framework that can be tailored and adapted to define a standard geared to the needs of a particular organisation.

Section 3 can be in different forms

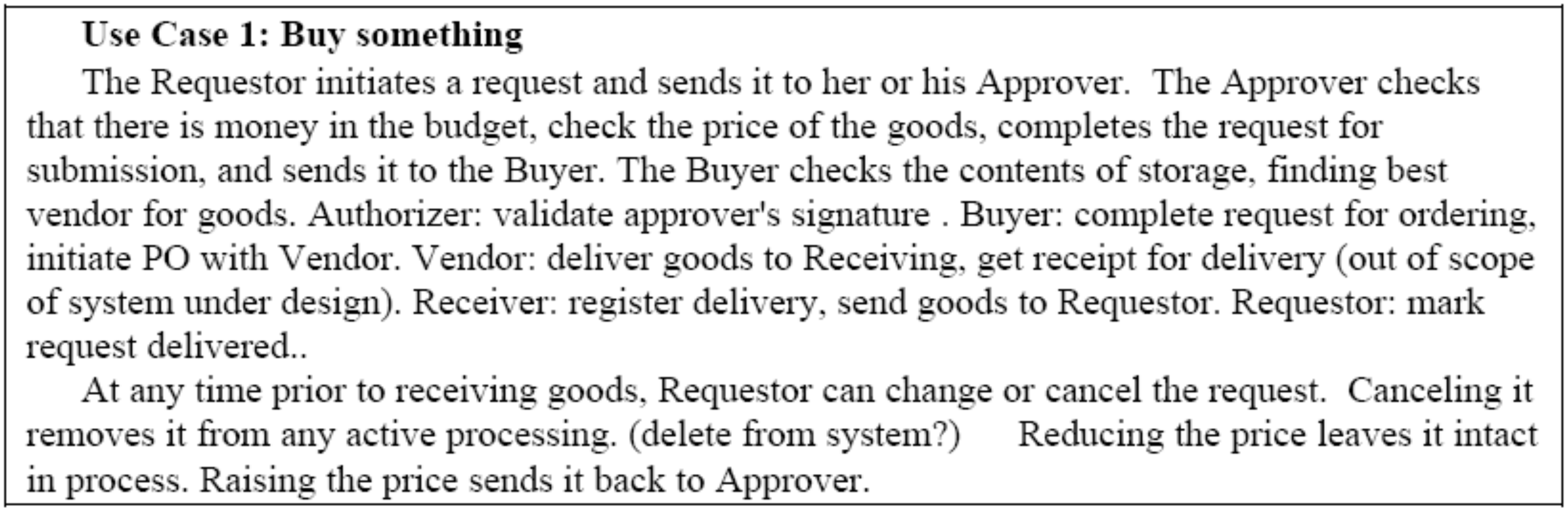
Specification: specifying functional requirements

- we will look at how to use “use cases” for specifying functional requirements
- a use case is
  - 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
    - just part of the SRS (see slide before)

Specification: use case formats

- 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

Specification: casual use case example



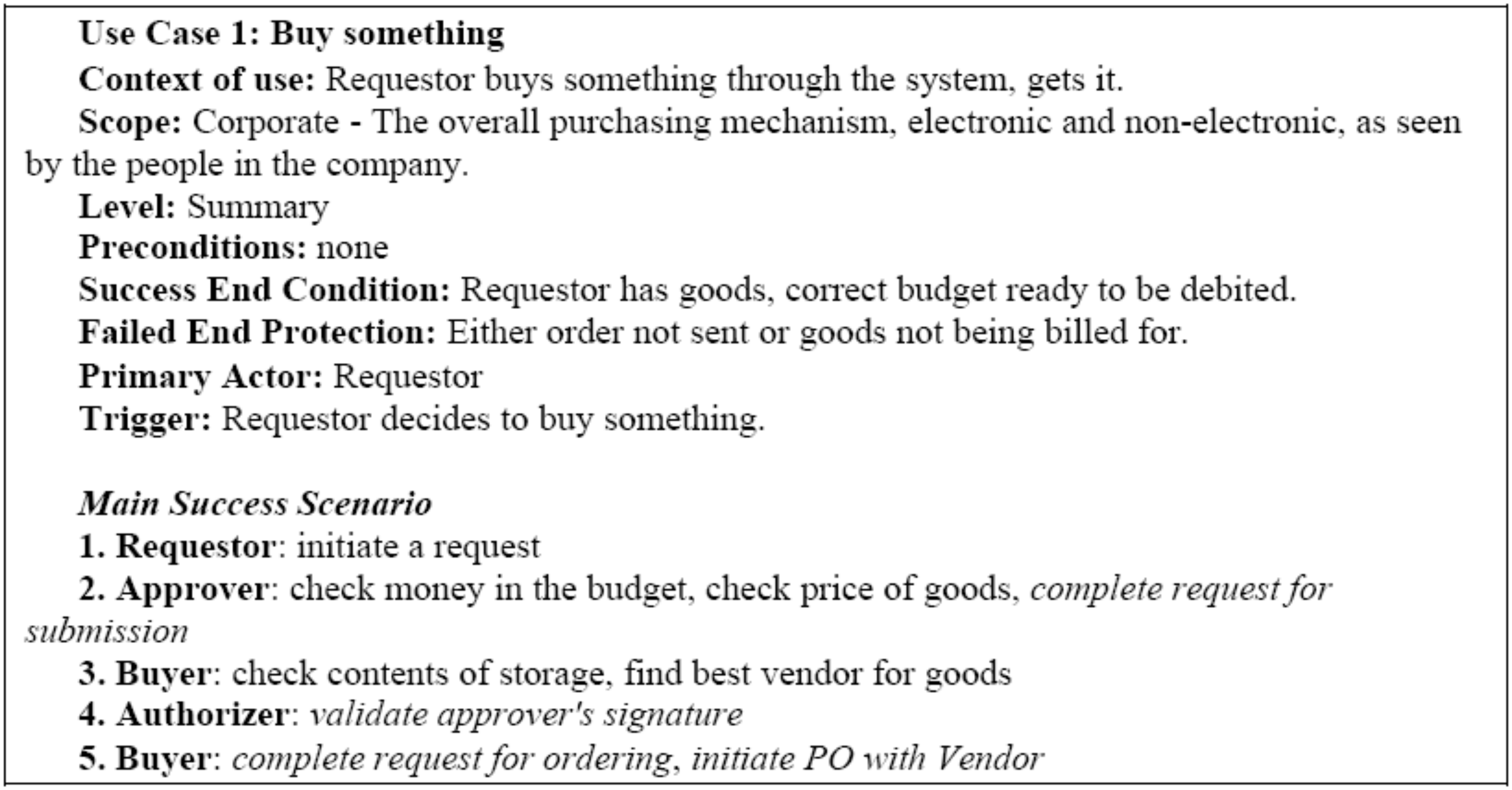
Use Case 1: Buy something

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.

Specification: 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*.

Deferred Variations
none

Project Information

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

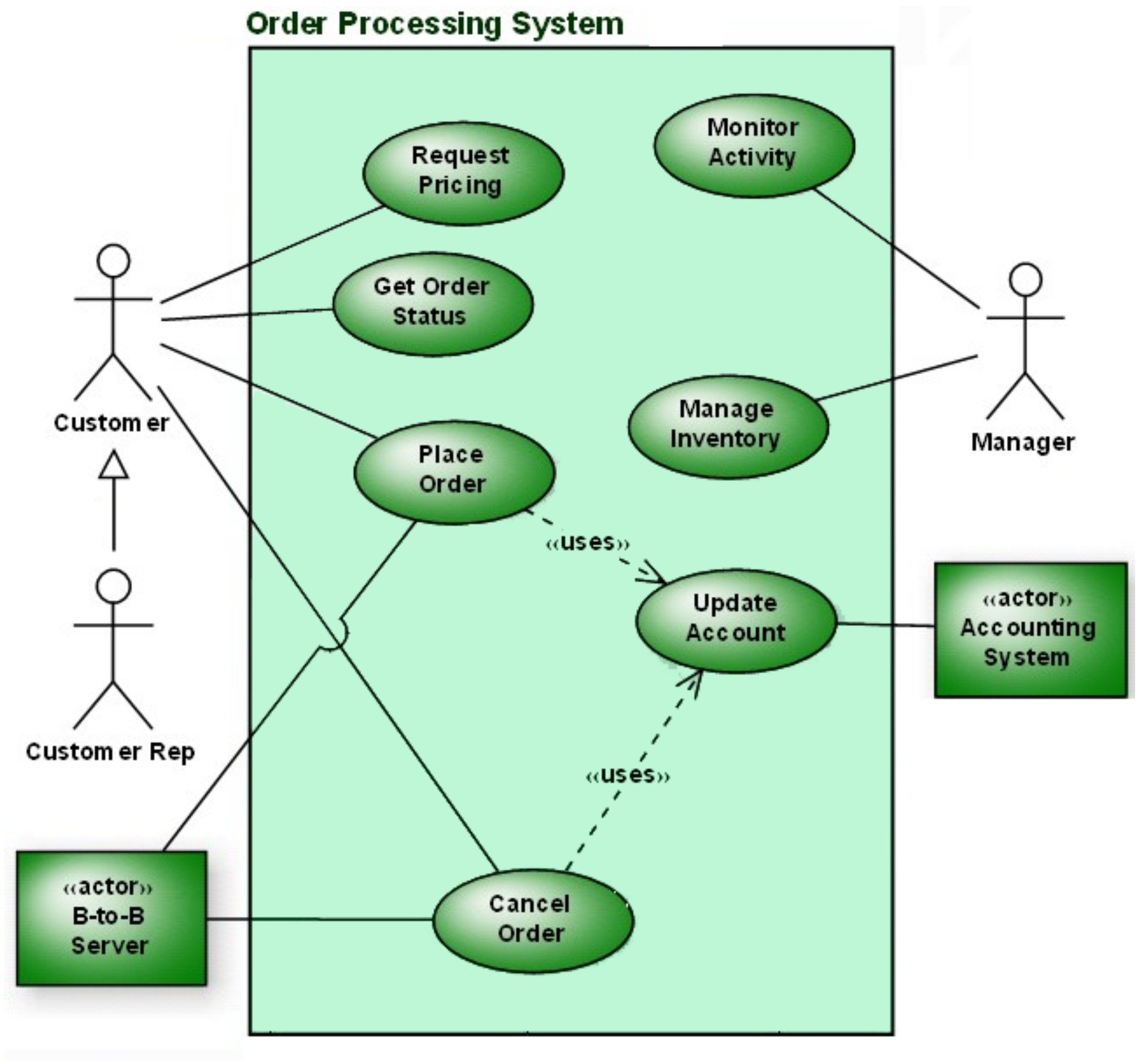
Calling Use Case: none
Subordinate Use Cases: see text
Channel to primary actor: Internet browser, mail system, or equivalent
Secondary Actors: Vendor
Channels to Secondary Actors: fax, phone, car
Open issues

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?

Specification: another use case example

http://epf.eclipse.org/wikis/openup/core.tech.common.extend_supp/guidances/examples/use_case_spec_CD5DD9B1.html

Specification: use case diagrams (aside)



Specification: use case diagrams (aside)

- 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 their internal behaviour
- 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

Specification: user story

- a user story is a short description of something your customer will do when they use your software focused on the value or result they will receive from doing this thing
- used a lot in agile development
- ~3 sentence description of what a software feature should do
- written in the customer's language
- should only provide enough detail to make a low-risk time estimate (a few days)
- Role-Goal-Benefit form:
    - "As a <ROLE>, I want to <GOAL> in order to <BENEFIT>"
    - As a student, I need to login to Piazza to finish the assignment

Specification: user story examples

- good (brief) examples:
  - 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

Specification: user story exercise #1

- let’s get the basic idea of use stories…
- in a group of 2…
  - write 2-4 user stories for Amazon
    - you should be writing 2-3 brief sentences in the form of “As a <ROLE>, I want to <GOAL> in order to <BENEFIT>”
- *Warning: I will be calling on some pairs to share…*

Specification: characteristics of good use stories

- good use cases follow INVEST:

  Independent
  Negotiable
  Valuable to users or customers
  Estimable
  Small
  Testable

Specification: 310 user story format (for now)

- for the user stories you will create for your assignment next week, you will follow a format that…
  - has 2-3 sentences of description
  - a number of tests that determine if the use case is satisfied

Specification: 310 format example #1

- As a company, I can use my credit card so I can pay for job postings
  - Notes: Accepts Visa, MasterCard, Amex. Consider: Discover
  - Test with Visa, MasterCard and Amex
  - Test with Diner's Club
  - Test with good, bad, and missing card ID numbers
  - Test with expired cards
  - Test with over $100 and under $100

Specification: 310 format example #2

- As a Creator, I want to upload a video from my local machine so that any user can view it.
    - Note: …
    - Test:
        - Click the “upload” button
        - Specify a video file to upload
            - Check that .flv, .mov. … extensions are supported
            - Check that files larger than 100MB results in an error
            - Check that movies longer than 10 min result in an error
        - Click “upload video” button
        - Check that progress is displayed in real time.

Specification: user story exercise #2

- let’s extend our basic idea of use cases…
- in a group of 2…
  - take your 2-4 user stories for Amazon from exercise #1 (or use the ones I provide)
    - add any notes
    - add tests
- *Warning: I will be calling on some pairs to share…*

Specification: user stories in agile development

- if the user story is a product backlog item also include
  - specifies acceptance or completion criteria that defines what is meant for this feature to be DONE
  - provide estimate of the required effort (story points)
- exercise on your own: take your Amazon user stories and try to prioritize them and add an effort estimate

Specification: what makes a good use story?

Independent

Negotiable

Valuable to Users or Purchasers

Estimable

Small

Testable

you are responsible for this material. Please read it and ask questions in the next lecture. (The material should be straightforward)

Specification: independent user stories

- little dependence between stories
- keeps development flexible
- makes estimation easier

Specification: negotiable user stories

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

Specification: valuable to purchasers or users

- Customer:
  - Purchaser: person who pays for the software
  - User: person who uses the software
- Make sure customer can estimate value of a story
- Not intended to be valuable only to developers
  - The backend database will be MySQL (no value to user, just restricts your options)

Specification: 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

Specification: 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 suggests break it up
- too small:
  - writing down the story may take longer than implementing it!
  - combine if too small

Specification: 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 two minutes)

Remember: Requirements activities

- Elicitation
- Analysis
- Specification
- Validation

Validation

Validation (for an SRS)

- A good requirement specification (SRS) must be
  - Correct
  - Complete
  - Unambiguous
  - Consistent
  - Ranked for importance and stability
  - Modifiable
  - Traceable

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

Validation: inspection

- Most common errors:
    - Omission: A requirement is missing
    - Inconsistency: Conflicting requirements
    - Incorrect facts
    - Ambiguity
- Inspection is the primary way to validate SRS
    - Should include various stakeholders (e.g., 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 areas not addressed in the SRS that need to be?
9. Are performance requirements stated?

Story name

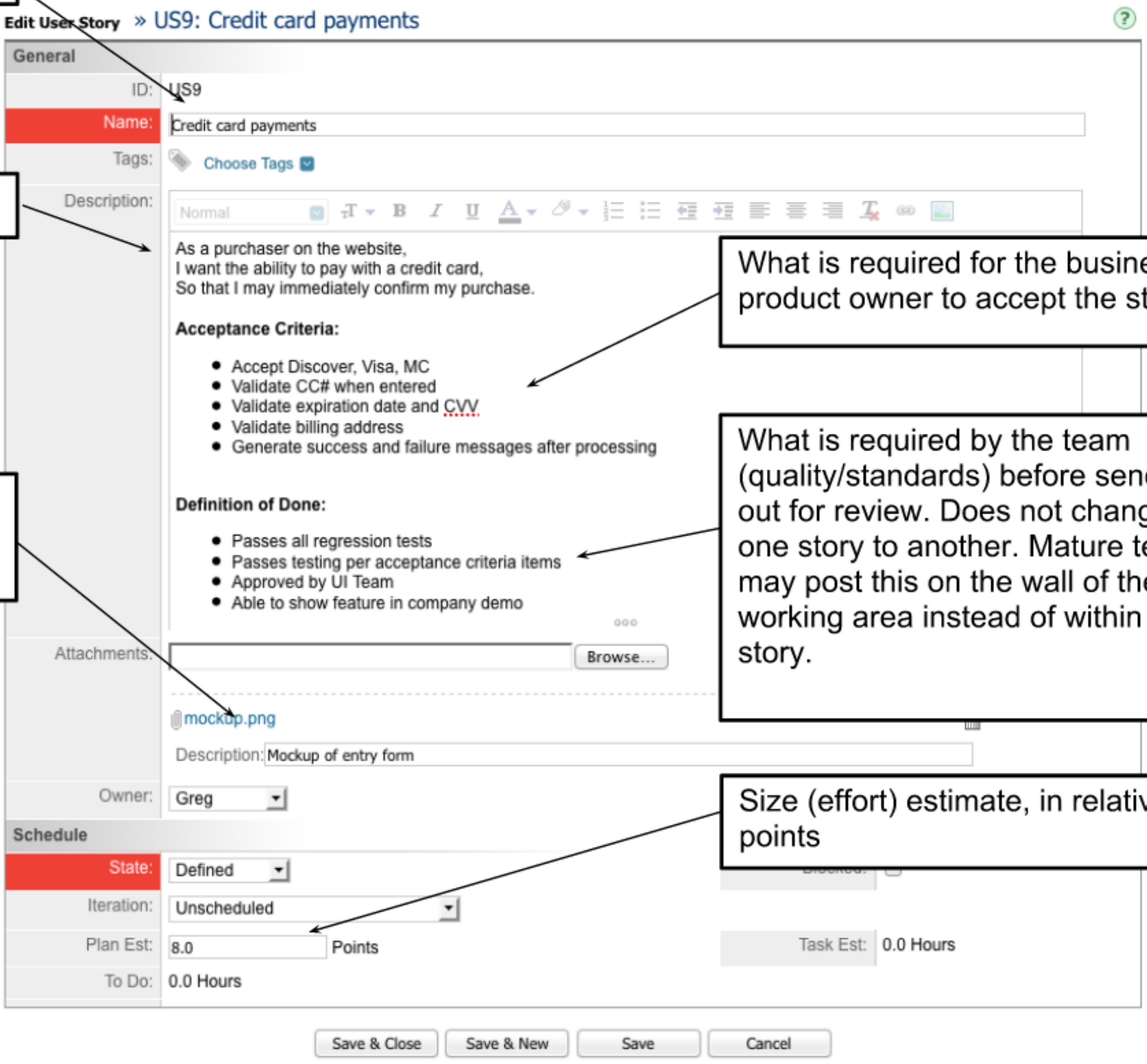
Value statement

Attach details and documents when necessary

What is required for the business and product owner to accept the story

What is required by the team (quality/standards) before sending out for review. Does not change from one story to another. Mature teams may post this on the wall of the team working area instead of within each story.

Size (effort) estimate, in relative points


Edit User Story » US9: Credit card payments

General

ID:	US9
Name:	Credit card payments
Tags:	Choose Tags

Description:

As a purchaser on the website,
I want the ability to pay with a credit card,
So that I may immediately confirm my purchase.

Acceptance Criteria:

- Accept Discover, Visa, MC
- Validate CC# when entered
- Validate expiration date and CVV
- Validate billing address
- Generate success and failure messages after processing

Definition of Done:

- Passes all regression tests
- Passes testing per acceptance criteria items
- Approved by UI Team
- Able to show feature in company demo

Attachments: mockup.png
Description: Mockup of entry form

Owner: Greg

Schedule

State:	Defined
Iteration:	Unscheduled
Plan Est:	8.0 Points
To Do:	0.0 Hours
Task Est:	0.0 Hours

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

Requirements Activity

(you need to hand in your work at end of class with names/ids on it)

1. Form large groups (6-8)
2. I’ll give you a sample system name and description.
3. Construct 3-6 simple user stories (no test info needed) to indicate the functionality you expect from the software you would build.
    - “As a <ROLE>, I want to <GOAL> in order to <BENEFIT>”

4. Send one person from your group to a group with a different #. DO NOT BRING YOUR USER STORIES WITH YOU!

5. The visitor is now a user and the group the visitor joined is now the development team. Using the interview techniques we've discussed, the development team needs to interview the user. Write 3-6 user stories about what you learn.

   DO NOT ASK THE USER DIRECTLY FOR THEIR USER STORIES!

6. Give the user their stories and send them back to their group. Discuss what comes back to your group.

7. Hand in the name of the system you first wrote user stories for, a name for the user stories that you wrote when the user visited you and all the names/ids of your group on a sheet and hand it in at the front.