CS189A - Capstone

Chandra Krintz
Department of Computer Science
UC Santa Barbara
http://www.cs.ucsb.edu/~ckrintz/
Today in CS189A

• Each lead overviews their project
  – Introduce self and team
  – 1-2 sentences what the project is about and what problems it will solve

• Requirements engineering and specification
  – Options for specification
    • Use cases
    • User stories – examples from Chandra’s project

• Team activities
  – Scrum (including PRD v1)
  – Each member: add 1 use case or user story (feature) to PRD v1; add to Product backlog trello board & break down into <1/2 day tasks w/ timings

• Section: Scrum, TA meetings, Sprint1 review & retrospective, refine 5+ use-cases/user-stories; Sprint 2 planning (share and show TA Sprint 2 trello board and burndown)
  – Sprint 2 = 11 days (starts Thursday)
Requirements engineering

• Process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
  – May range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.
  – Precisely stated, unambiguous

• User requirements
  – Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

• System requirements
  – A structured document setting out detailed descriptions of the system’s functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor
User and system requirements

User requirement definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20 mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.
Be Careful About Ambiguities in Informal Specifications

• “The input can be typed or selected from the menu“
  – The input can be typed or selected from the menu or both
  – The input can be typed or selected from the menu but not both

• “The number of songs selected should be less than 10”
  – Is it strictly less than?
  – Or, is it less than or equal?

• “The user has to select the options A and B or C”
  – Is it “(A and B) or C”
  – Or, is it “A and (B or C)”
Functional and Non-functional Requirements

• Functional requirements (user + system requirements)
  – Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  – May also state what the system should not do.

• Domain requirements
  – Constraints on the system from the domain of operation
    • Operating environment (e.g. underwater, temp range, environmental conditions to be tolerated)

• Non-functional requirements
  – Constraints on services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  – Often apply to the system as a whole rather than individual features or services.
Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.

- Process requirements may also be specified mandating a particular IDE, programming language or development method.

- Non-functional requirements may be more critical than functional requirements and effect overall architecture (e.g. minimize communications). If not met, system may be useless.

- A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.
  - It may also generate requirements that restrict existing requirements
Types of Non-functional Requirements

- Non-functional requirements
  - Product requirements
    - Efficiency requirements
    - Dependability requirements
    - Security requirements
  - Organizational requirements
    - Environmental requirements
    - Operational requirements
    - Development requirements
    - Legislative requirements
  - External requirements
    - Accounting requirements
    - Safety/security requirements
  - Usability requirements
  - Performance requirements
  - Space requirements
  - Regulatory requirements
  - Ethical requirements
  - Safety/security requirements
  - Accounting requirements
  - Operational requirements
  - Development requirements
  - Legislative requirements
  - Environmental requirements
  - Operational requirements
  - Development requirements
  - Legislative requirements
## Metrics for Specifying Non-functional Requirements

<table>
<thead>
<tr>
<th>Property</th>
<th>Measure</th>
</tr>
</thead>
</table>
| Speed          | Processed transactions/second  
User/event response time  
Screen refresh time |
| Size           | Mbytes  
Number of ROM chips |
| Ease of use    | Training time  
Number of help frames |
| Reliability    | Mean time to failure  
Probability of unavailability  
Rate of failure occurrence  
Availability |
| Robustness     | Time to restart after failure  
Percentage of events causing failure  
Probability of data corruption on failure |
| Portability    | Percentage of target dependent statements  
Number of target systems |
PRD: the product requirements document

• The official statement of what is required of the system developers
• Includes a specification of both user and system requirements

• Defines **WHAT the system should do** not **HOW it should do it**
  – Design comes later

• Agile and extreme SWE processes express requirements as
  – **Use cases** – how a system will act
  – Or as scenarios called **user stories** (describe result/benefit of it)
Agile Requirements Specification

1. Define project specifics
2. Team goals and objectives
3. Background and strategic fit
4. Assumptions
5. User Stories or Use Cases
6. User Interaction and Design
7. Questions
8. What we’re NOT Doing

- Evolve the document over time, concurrently with development

Required reading: https://www.atlassian.com/agile/requirements
Requirements validation techniques

• Requirements reviews
  – Systematic manual analysis of the requirements.
  – Review/commit changes to repository as part of workflow
    • Multiple team members OK it before committing
    • All team members get notification when its updated

• Prototyping
  – Using an executable model of the system to check requirements.

• Test-case generation
  – Developing tests for requirements to check testability.
  – Your test cases / acceptance tests should be github commits
Use Cases

• Use cases document the behavior of the system from users’ point of view.
  – By user we mean anything external to the system
  – Consist of:
    actors – scope – goals – steps – success

• An actor is a role played by an outside entity that interacts directly with the system
  – An actor can be a human, or a machine or program
  – Actors are shown as stick figures in use case diagrams

Customer
Use Cases

- A **use case** describes the possible **sequences of interactions** among the system and one or more actors in response to some initial stimulus by one of the actors
  - Each way of using the system is called a use case
    - Sequence of interactions
      - A use case is not a single scenario but rather **a description of a set of scenarios**
      - For example: *Creating an account* or *Performing transaction* or *Applying for a loan*

- In a use case, the system is considered **a black-box**.

*We are only interested in describing externally visible behavior*
Use Cases

• To define a use case, group all transactions that are similar

• A typical use case might include a main case, with alternatives taken in various combinations and including all possible exceptions that can arise in handling them

  – Use case for an online banking app: *Performing a Transaction*
    • Subcases could include *Making Deposits, Making Withdrawals*, etc., together with exceptions such as *Overdrawn* or *Account Closed*
  
  – *Apply for a Loan* could be a separate use case since it is likely to involve very different interactions
Generalization in Use Case Diagrams

UC Santa Barbara

- Customer
  - Individual Customer
  - Corporate Customer

- Validate User
  - Check Password
  - Retinal Scan

Indicates generalization
Use Cases

• Description of a use case should include **events exchanged between objects and the operations performed by the system that are visible to actors**

• **Have preconditions and postconditions**
  – Precondition states all assumptions about state/environment of system that impacts the actor(s) in this use case
  
  – Postcondition is an acceptance test (how to know when implementation is complete) and describes externally visible state/environmental changes
Use Cases

• To define a use case, group all transactions that are similar in nature.
• A typical use case might include a main case, with alternatives taken in various combinations and including all possible exceptions that can arise in handling them.
  – Use case for a bank: *Performing a Transaction at the Counter*
    • Subcases could include *Making Deposits*, *Making Withdrawals*, etc., together with exceptions such as *Overdrawn* or *Account Closed*.
  – *Apply for a Loan* could be a separate use case since it is likely to involve very different interactions.
• Description of a use case should include **events exchanged between objects and the operations performed by the system that are visible to actors**.
• **Have preconditions and postconditions**
  – Precondition states all assumptions about state/environment of system that impacts this use case.
  – Postcondition as an acceptance test (how to know when implementation is complete).
Use case: Update Benefits


Precondition: Employee has logged on to the system and selected “update benefits” option

Flow of Events:

Basic Path:
1. System retrieves employee account from Employee Account Database
2. System asks employee to select medical plan type; uses Update Medical Plan
3. System asks employee to select dental plan type; uses Update Dental Plan
...

Alternative Paths:
If health plan is not available in the Employee’s area the employee is informed and asked to select another plan (exceptional cases that must be handled)
Employee selects cancel, logs out, or leaves page at any point prior to confirming the update (an end-early path)

Postcondition: Employee account plan type has been updated in the Employee Account Database or nothing has changed (end-early paths)

Note that code tests can be written for pre/post conditions
User Stories

• Similar to Use Cases but not the same
  – User stories are centered on the result and the benefit of the thing you’re describing, whereas use cases are more granular, and describe how your system will act.  From: http://www.boost.co.nz/blog/2012/01/use-cases-or-user-stories/

• Use cases: actors – scope – goals – steps – success
  – Details of most important requirements worked out ahead of time to ensure that everyone is on the same page
  – Useful for groups of similar stories and describing overall system
    • Use cases decompose stories into actions in the system

• User stories: scope of a feature + acceptance criteria
  – Each feature is captured as a story; stories easily prioritized
  – A story is a place holder for discussion and planning poker in a sprint

See recommended reading links for examples and suggestions
User Stories

• Stems from Behavior Driven Development (BDD)
  – Employed in XP/Agile processes
  – Improves communication/understanding of requirements by all involved

• An outside-in methodology
  – Encourage discovery: drill down on a feature set to achieve desired (business) outcomes

• See for examples
  – Dan North: “What’s in a Story?”
  – Agile Modeling: “Introduction to User Stories”
Writing Good User Stories

- It's typically difficult to get started writing good user stories
  - Here are 4 steps to make it easier

1. As a [role], I can [feature] so that [reason]
2. Use index cards and a sharpie
3. Make it testable with acceptance criteria or demo plan
4. Connect the dots

From: http://codesqueeze.com/the-easy-way-to-writing-good-user-stories/
As a [role], I can [feature] so that [reason]

- Role – a person; feature – something your project does; reason – a solution to a problem the person has
  - This is a pattern that is commonly used for stories

  As a account owner, I can check my balance online so that I can access my daily balance 24 hours a day.

- Variations
  - As a [role], I want [feature] because [reason]
  - As a [role], I can [feature]
  - As a [role], I can [feature] so that [reason]
Use index cards and a sharpie

• Although there is software out there to help you with this
  – Jira, Trello, Pivotal tracker

• Physically writing out stories facilitates keeping the story clear, concise, and of the appropriate size
  – Keep them short and sweet and unambiguous
    • Goal is to aid communication, not overly detailed or long-winded
  – It also enables you to doodle/draw the outline of the user interface

• If it doesn’t fit, break up the story into sub-stories
Make it testable with acceptance test or demo

• If they are short and sweet and without detail, how do we know when they are “done”?

Story: As a [role], I can [feature] so that [reason]

• Include an acceptance test (what to demo when done):

  Scenario 1: Title
  Given [context]
  And [some more context]…
  When [event]
  Then [outcome]
  And [another outcome]…

Example

  Scenario 1: Account balance is negative
  Given the account’s balance is below 0
  And there is not a scheduled direct deposit that day
  When the account owner attempts to withdraw money
  Then the bank will deny it
  And send the account owner a nasty letter.

• All tests should fit on back of story card (in sharpie)
  – If they don’t, break up the story into two
  – You should be able to code them in a few lines of code
Writing Good User Stories

• It's typically difficult to get started writing good user stories
  – Here are 4 steps to make it easier

1. As a [role], I can [feature] so that [reason]
2. Use index cards and a sharpie
3. Make it testable with acceptance criteria or demo plan
4. Connect the dots
  – Lay the stories out, determine which ones are dependent on others, prioritize them in order to provide a working system/product each sprint

From: http://codesqueeze.com/the-easy-way-to-writing-good-user-stories/
Chandra’s Example Project: IoT Sensor Data Viewer

1. As a user, I can login, so that I can use the system
   - Login: Given a user name and password, when saved on a web page form, then the user is logged in and can access viewer services (test = a test page is loaded that is only accessible to logged in users)

2. As a sensor, I can upload my data to the viewer over the Internet, so that it is persisted there
   - Sensor Upload: Given a sensor connected to the Internet, when a script invokes an upload API, then a window of data from the sensor is uploaded to the viewer and the viewer saves the data (test = see the data locally, run the script, see the on the viewer)

• Note that there is no notion of “how” to persist: database, file, etc
• That can be included or pushed off until later…
Chandra’s Example Project: IoT Sensor Data Viewer

- As a [role], I can [feature] so that [reason]

  Title: Given [context], when [event], Then [outcome]

3. As a developer, I can save my environment to DockerHub, so that I download and use it on different machines
   - DockerHub: Given a container, when exited and pushed to DockerHub, then the container can be downloaded and run on a different machine (test = upload, download, and run of container)

4. As an API request, I can store incoming data to a database, so that it is persisted in a structured format
   - Valid DB Storage Request: Given a API request over the Internet, when a valid request occurs, a window of data is stored in a DB (test1 = make an valid request with data: view data separately in DB, view schema)

   - Invalid DB Storage Request: Given a API request over the Internet, when an invalid request occurs, an error is logged/returned (test2 = make an invalid request: no change to DB, throw error message)
Chandra’s Example Project: IoT Sensor Data Viewer

• As a [role], I can [feature] so that [reason]
  Title: Given [context], when [event], Then [outcome]

5. As an API, I can handle multiple, concurrent requests at once, so that throughput is maximized given available server resources
  – Throughput: Given an API, when multiple requests come in, multiple threads handle the requests, \#threads = VALxCPUs (test = measure throughput for different numbers of threads and CPU resources)

• Others:
  – As a user, I can access API operation, so that … //define/implement different API functions
  – As a user, I can access multiple web pages for the service, so that… //define/implement UI and/or front end
  – As a user, I can view sensor data over the Internet, so that… //add Chart.js for fake, static data
  – As a user, I can view dynamic sensor data over the Internet, so that… //add Chart.js + GraphQL for fake, dynamic data
  – Next: connect vis to DB, add API ops, add edge server support (revisit earlier features), handle errors if/when sensors go down or send bad data, support multiple vis options, add data analysis/ML algorithms on data for extracting insights from data, …
Today in CS189A

- Each lead overviews their project
  - Introduce self and team
  - 1-2 sentences what the project is about and what problems it will solve

- Requirements engineering and specification
  - Options for specification
    - Use cases
    - User stories—examples from Chandra’s project

- Team activities
  - Scrum (including PRD v1)
  - Each member: add 1 use case or user story (feature) to PRD v1; add to Product backlog trello board & break down into <1/2 day tasks w/ timings

- Section: Scrum, TA meetings, Sprint1 review & retrospective, refine 5+ use-cases/user-stories; Sprint 2 planning (share and show TA Sprint 2 trello board and burndown)
PRDv1: Your **Living** Requirements Document: A Shared Google Doc (due in <1 week)

- **Authors, Team, Project Title**
- **Intro** – including problem, innovation, science, core technical advance (2-3 pages)
  - Define project specifics, team goals/objectives, background, and assumptions
- **System architecture overview**
  - High level diagram (1 page)
  - User interaction and design (1+ page)
- **Requirements (functional and non-functional)**
  - User stories or use cases (links) → 10 for PRDv1 prioritized
  - Prototyping code, tests, metrics (5+ user stories): [github commits/issues]
- **System models:** contexts, sequences, behavioral/UML, state
- **Appendices**
  - Technologies employed
PRDv2: Your Living Requirements Document: A Shared Google Doc (due 1 month after PRDv1)

- Authors, Team, Project Title
- Intro: problem, innovation, science, core technical advance
  - Define project specifics, team goals/objectives, background, and assumptions
- System architecture overview
  - High level diagram (1 page)
  - User interaction and design (1 page)
- Requirements (functional and non-functional)
  - User stories or use cases (links) → 20+ for PRDv2 prioritized
  - Prototyping code, tests, metrics (10+ user stories): github commits/issues
- System models (1+ pages)
  - Contexts, interactions, structural, behavioral (UML)
  - Use cases, sequencing, event response, system state, classes/objects
- Appendices - Technologies employed