CS189A
Software Engineering: Concepts and Practices

Jianwen Su
Department of Computer Science
UC Santa Barbara

https://capstone.cs.ucsb.edu/cs189a/cs189a_sched.html
Today’s Lecture

- Team summaries: Project goals and milestones
  AgMonitor, Alcon, Appfolio, Bill, Invoca, NavAir, NavSea, Teladoc, WellHealth, Xenon
  - Introduce self and team
  - What the project is about and what problems it will solve

- Management of software development
- Ethics

- Fall quarter plan
- Requirements engineering and specification
  - Use cases

- Team meetings
Project Management

- Getting started
- During the development cycle:
  - Progress
  - Issue tracking (technical)
  - Reflect and planning (sprint retro and plan)
  - Management issues:
    mentor communication,
    team members ad hoc situations
    possibly others
Ethics

- NDA: is binding if you signed
  - Read carefully and understand what you are (not) permitted to do
- Your system should respect privacy laws (e.g. client info)
- Being truthful
- Working as a team
Overall Plan @Week 4

- Four 2-week sprints (dates adjusted)
  - Oct 11-25 (PRD v1 – tools, technologies, design, terminology);
  - Oct 25-Nov 8 (use cases/user studies, prototyping, PRD v1);
  - Nov 8-22 (design, prototyping, testing, PRD v2);
  - Nov 22-Dec 3 (prototype demo/pres prep, prototyping and testing)

- Fall presentations and demos: Dec 3, 3:30-5:30 (tentative)

- Specify what the product will do
  - Vision statement
  - Product Requirements Document (PRD) (due Oct 29 and Nov 26* or 29)
  - Design tools, brainstorming, coding (tests and implementation)

- Build and test an initial prototype
  - Typically teams iterate on these activities until they converge to a working prototype!
Key Dates Illustrated

- Four 2-week sprints (dates adjusted)
  - Oct 11-25 (PRD v1 – tools, technologies, design, terminology);
  - Oct 25-Nov 8 (use cases/user studies, prototyping, PRD v1);
  - Nov 8-22 (design, prototyping, testing, PRD v2);
  - Nov 22-Dec 3 (prototype demo/pres prep, prototyping and testing)

- Fall presentations and demos: Dec 3, 3:30-5:30 (tentative)
This Week’s Plan

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

- Section: Scrum, Sprint 1:

Upcoming deadlines:

- Oct 25 (next lecture)
  - 1-slide team report (3 minutes)
    - Sprint 1 review/retro; PRD progress update
  - Team meetings: Sprint 2 planning

- Oct 29: Product Requirement Document (release 1)
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: Examples

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.
Caution: 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 vs 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

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

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

- **Both** document how the system responds from an *external* perspective (when viewed from the *outside*) – like a black box…

So - we are only interested in describing externally visible behavior
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
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.
Generalization in Use Case Diagrams

Indicates generalization
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*
    
    o 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
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 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
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 an 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
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]…

- 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

- Its 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/
PRDv1: Your Living Requirements Document: A Shared Google Doc (due in ~2 weeks)

- 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