CS189A - Capstone

Chandra Krintz
Department of Computer Science
UC Santa Barbara
http://www.cs.ucsb.edu/~ckrintz/
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 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
  – 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
- Usability requirements
  - Performance requirements
  - Space requirements

Organizational requirements
- Dependability requirements
- Environmental requirements
- Operational requirements
- Development requirements
  - Accounting requirements
  - Safety/security requirements

External requirements
- Security requirements
- Regulatory requirements
- Ethical 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 User/event response time Screen refresh time</td>
</tr>
<tr>
<td>Size</td>
<td>Mbytes Number of ROM chips</td>
</tr>
<tr>
<td>Ease of use</td>
<td>Training time Number of help frames</td>
</tr>
<tr>
<td>Reliability</td>
<td>Mean time to failure Probability of unavailability Rate of failure occurrence Availability</td>
</tr>
<tr>
<td>Robustness</td>
<td>Time to restart after failure Percentage of events causing failure Probability of data corruption on failure</td>
</tr>
<tr>
<td>Portability</td>
<td>Percentage of target dependent statements Number of target systems</td>
</tr>
</tbody>
</table>
The software requirements document

• The software requirements document is the official statement of what is required of the system developers.
• Should include both a definition of user requirements and a specification of the system requirements.
• Defines **WHAT the system should do rather than HOW it should do it.**
  – Design comes later

• Agile and extreme SWE processes express requirements as
  – **Use cases**
  – Or as scenarios called **user stories**
Use Cases

- Use cases document the behavior of the system from users’ point of view.
  - By user we mean anything external to the system
- 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
  – A use case is not a single scenario but rather a description of a *set of scenarios*
  – For example: *Creating an account*
  – Individual use cases are shown as named ovals in use case diagrams

  ![Creating an account](image)

• A use case involves a sequence of interactions between the initiator and the system, possibly involving other actors.
• In a use case, the system is considered a black-box.

*We are only interested in describing externally visible behavior*

*Have preconditions and postconditions*
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

• 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
Generalization in Use Case Diagrams

UC Santa Barbara

Validate User

Check Password

Retinal Scan

Indicates generalization

Customer

Individual Customer

Corporate Customer
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

Postcondition: Employee account plan type has been updated

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
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 stories
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 keep a 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 stories

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

- Include an acceptance test:

  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
Connect The Dots

• Most software teams lump together the terms **Analysis and Design** into one pre-development phase.
  – This is unfortunate because with enough effort in first determining what you wish to build (aka Analysis) it is possible to understand the domain enough not to need up front architecture (aka Design).

• Take all of your user stories and lay them out on a table/floor
  – Give each a number
  – Identify on paper, which stories are connected, related, dependent
    • Did you miss some? Similar to use case interrelationships
  – Goal of design/prototyping: connect the minimum sufficient to understand the full picture/project (is this possible? If not, why?)
    • Work to design/prototype these stories first
  – Prioritize (mandatory, important, desirable), and order
Requirements checking

- **Validity.** Does the system provide the functions which best support the customer’s needs?
- **Consistency.** Are there any requirements conflicts?
- **Completeness.** Are all functions required by the customer included?
- **Realism.** Can the requirements be implemented given available budget and technology?
- **Verifiability.** Can the requirements be checked?
Requirements validation techniques

• Requirements reviews
  – Systematic manual analysis of the requirements.

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

• Test-case generation
  – Developing tests for requirements to check testability.
Your Living Requirements Document: A Shared Google Doc

- Authors, Team, Project Title
- Intro – including problem, innovation, science, core technical advance
- Glossary of Terms
- System architecture overview
- Requirements (functional and non-functional)
- System models
- Appendices
  - Technologies employed
Draft 1: Your Living Requirements Document: A Shared Google Doc

- Authors, Team, Project Title
- Intro – including problem, innovation, science, core technical advance
- Glossary of Terms
- System architecture overview
  - High level picture
- Requirements (functional and non-functional)
  - At least 10 use cases or user stories -- prioritized
- Prototyping code and test cases (Github URL)
- System models
  - N/A
- Appendices
  - Technologies employed

- Authors, Team, Project Title
- Intro – including problem, innovation, science, core technical advance
- Glossary of Terms
- System architecture overview
  - SW architecture with significant detail; components; APIs; dependencies
- Requirements (functional and non-functional)
  - Update original use cases or user stories as needed - reprioritize
  - An additional 20 use cases or user stories -- prioritized
- Prototyping code and test cases (Github URL)
- System models (design)
  - Contexts, interactions, structural, behavioral (UML)
  - Use cases, sequencing, event response, system state, classes/objects
- Appendices -- Technologies employed