CS 307: Software Engineering

Lecture 5: Requirements Analysis

Prof. Jeff Turkstra
Announcements

- **Project Charter** due today.
  Friday, January 20 3:00pm

- **Homework 1**
  - Due Monday, January 23 11:59pm

- **Code repository, project name, and product backlog**
  - Friday, January 27
Lecture 05

- Software Engineering Projects
- Problems vs. Solutions
- Requirements or constraints
- Requirements gathering
Software engineering projects

- Most projects are evolutionary or maintenance projects, involving work on legacy systems
  - Corrective: fixing defects
  - Adaptive: changing the product in response to changes in...
    - The system in which it operates
    - Available information
    - Organizational changes
  - Enhancement: adding new features
  - Reengineering or perfective: changing the system internally to be more maintainable
Greenfield vs. Brownfield

- **Greenfield:** “from scratch”
  - New development
  - No constraints from prior work
  - Construction on “greenfield” land where there is no need to demolish or remodel
  - Minority of projects

- **Brownfield:** must work with existing systems
  - Take into account and coexist with already-existing software and environments
Requirements

- Domain analysis
- Defining the problem
- Requirements gathering
  - Obtaining input from as many sources as possible
- Requirements analysis
  - Organizing this information
Design

Deciding how the **requirements** should be implemented using available technology

**Includes**

- **Systems engineering**: deciding what should be in hardware and what in software
- **Software architecture**: dividing the system into subsystems and deciding how they will interact
- **Detailed design of the internals**
- **User interface design**
- **Design of databases**
- etc
Next

- Modeling
  - Creating representations of the domain and software
    - User stories, use cases
- Programming
- Quality assurance
  - Reviews and inspections
  - Testing
- Deployment
- Managing the process
The dilemma

“I know you believe you understood what you think I said, but I am not sure you realize that what you heard is not what I meant...”
Domain analysis

- Process by which software engineer learns about the domain to better understand the problem
  - The **domain** is the general field of business or technology in which the software will be used
  - A **domain expert** is a person who has deep knowledge of the domain

- **Benefits**
  - Faster development
  - Better system
  - Easier to anticipate future modifications
Domain analysis document

A) Introduction
B) Glossary
C) General knowledge about the domain
D) Customers and users
E) The environment
F) Tasks and procedures currently performed
G) Competing software
H) Similarities to other domains
Defining the problem

- A problem can be expressed as:
  - A difficulty the users or customers are facing
  - An opportunity that will result in some benefit (improved productivity, sales, etc)
- Solution will normally entail developing software
- Good problem statements are short and succinct
Collect the requirements

- **Real – functional and performance**
  
The furnace temperature alarm indicator must be set any time the furnace temperature is not ±5°C of the set point for more than 10 seconds

- **Constraints – non-functional**
  
  All software will be written in ANSI C for the 68040
- **Possible** – non-measurable, subjective, political
  
The system will be user friendly.
  The system must be designed in Senator X’s state

- **Probably not** – expectations, wishes, desires
  
We must be able to sell at least 10,000 copies
Gathering and analyzing requirements

- **Observation**
  - Read documents and discuss requirements with users
  - Shadow important potential users as they do their work
    - Ask the user to explain what he or she is doing
  - Session video recording
Interviewing

- Ask about specific details
- Ask about the stakeholder’s vision of the future
- Ask for alternative ideas
- Ask for other sources of information
- Ask stakeholder to draw diagrams
Prototype

- Draw pictures, show them to users
- Develop a mock-up UI
  - May be written in a rapid prototyping language
  - Does not normally perform computations, interact with databases or other systems
- Maybe only a particular aspect of the system
Explore the requirements

- Need to determine what the system is
- What the rest of the world looks like
- The boundary (fuzzy) between the two
The world

System
Comments

- If you can change it, then it probably is in the system
- If you can influence it, then it probably forms a part of the boundary
- If you cannot change it, then it probably belongs to the rest of the world
Understand the requirements

- **Measure** – how will you know when you have met the requirements?
- **Risks** – how certain are you that you can meet the requirements?
- **Constraints** – should the solution space be limited?
Organize and prioritize

- Must have and can be met?
- Risks – must have but not sure can be met?
  - Buy information
  - Build prototypes
- Nice to have?
  - How well can you “live” without it?
    With some pain
- Gold plating
  - Not much or no pain
Generate the requirements

- List the requirements indicating priority
- For Scrum, these should be user stories in the product backlog
  - Add and modify throughout the semester
  - Create as many as you want, even if there isn’t enough time to finish all of them
- Establish a requirement tracking procedure
- Establish a requirements verification plan
User stories and Use Cases

- **User Stories** are short, simple descriptions of a feature told from the perspective of the person who wants the feature
  - Usually a user or customer
  
  “As a <type of user>, I want <some goal>”

- **Use Cases** are an expansion of a User Story

<table>
<thead>
<tr>
<th>Actor actions</th>
<th>System responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Click on a hotel logo</td>
<td>2. Display hotel details</td>
</tr>
<tr>
<td>3. Click “Book Now”</td>
<td>4. Display payment form</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Choosing user stories

- Often one user story (or a small number) can be identified as central to the system
  - The entire system can be built around these user stories
- Other reasons to focus on particular user stories:
  - Some may represent a high risk (problematic implementation)
  - Some may have high political or commercial value
User stories can...  

- Help define the **scope** of the system  
- Be used to **plan** the development process  
- Be used to **develop and validate** requirements  
- Form the basis for **test cases**  
- Help structure user documentation
User stories are not a panacea

- They must be validated
  - Using requirements validation methods
- Some aspects are not covered by user stories
  - E.g., internal details
Use case analysis

- A use case is a typical sequence of actions that a user performs to complete a given task

- Use case model
  - Set of use cases
  - Optional description or diagram indicating how they are related
Use case

- Cover the full sequence of steps from beginning to end
- Describe the user’s interaction with the system
  - Not the computations performed
- As independent as possible from the UI design
- Only include actions arising from actor interacting with system
Use case diagrams

- Use Case
  - Horizontal ellipse
- Actor
  - Stick figure
  - Person, organization, or external system
- Associations
  - Lines
  - Connect actors and use cases
  - Can use an arrow to show initial invocation, but may be confusing
Extensions

- Make **optional** interactions explicit
- Handle **exceptional** cases
- Keeps basic use case simple
- Think “hardware interrupt”
  - Don’t know when it will be invoked
  - Or if it will at all
Generalizations

- Similar to superclasses in a class diagram
- Represents several similar use cases
- One or more specializations provides details of the similar use cases
Inclusions

- Allow one to express commonality between several use cases
- Included in other use cases
  - Even very different use cases can share a sequence of actions
  - Avoids repeating details in multiple use cases
- Represent the performing of a lower-level task with a lower-level goal
Example use case

Use case: Open file

Related use cases:
Generalization of:
- Open file by typing name
- Open file by browsing

Steps:

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose “Open...” command</td>
<td>2. File open dialog appears</td>
</tr>
<tr>
<td>3. Specify filename</td>
<td>5. Dialog disappears</td>
</tr>
<tr>
<td>4. Confirm selection</td>
<td></td>
</tr>
</tbody>
</table>
Use case: Open file by typing name

Related use cases: Specialization of: Open file

Steps:

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose “Open...” command</td>
<td>2. File open dialog appears</td>
</tr>
<tr>
<td>3a. Select text field</td>
<td></td>
</tr>
<tr>
<td>3b. Type file name</td>
<td>5. Dialog disappears</td>
</tr>
<tr>
<td>4. Click ‘Open’</td>
<td></td>
</tr>
</tbody>
</table>
Use case: **Open file by browsing**

**Related use cases:**
Specialization of: Open file
Includes: Browse for file

**Steps:**

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose “Open...” command</td>
<td>2. File open dialog appears</td>
</tr>
<tr>
<td>3. Browse for file</td>
<td>5. Dialog disappears</td>
</tr>
<tr>
<td>4. Confirm selection</td>
<td></td>
</tr>
</tbody>
</table>
Use case: Attempt to open a nonexistent file

Related use cases:
Extension of: Open file by typing name

Steps:

<table>
<thead>
<tr>
<th>Actor Actions</th>
<th>System responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose “Open...” command</td>
<td>2. File open dialog appears</td>
</tr>
<tr>
<td>3a. Select text field</td>
<td></td>
</tr>
<tr>
<td>3b. Type file name</td>
<td></td>
</tr>
<tr>
<td>4. Click ‘Open’</td>
<td>5. System indicates file does not exist</td>
</tr>
<tr>
<td>6. Correct the file name</td>
<td></td>
</tr>
<tr>
<td>7. Click ‘Open’</td>
<td>8. Dialog disappears</td>
</tr>
</tbody>
</table>
Use case: **Browse for file (inclusion)**

**Steps:**

**Actor Actions**
- 1. If the desired file is not displayed, select a directory.
- 2. Repeat step 1 until desired file is displayed.

**System responses**
- 2. Contents of directory is displayed.
Manage the requirements

- Review them
- Track them
- Verify them
- Control the changes to them
Remember

- The purpose of exploring requirements is to reduce the uncertainty about
  - what was needed and not requested
  - what was asked for and not needed
Problems

- Unknown – missing requirements
- Incomplete requirements
- Ambiguous requirements
- Non-requirements
Reviewing requirements

- Each requirement should...
  - Have benefits that outweigh the costs
  - Be important for the solution
  - Be unambiguous
  - Be logically consistent
  - Lead to a quality system
  - Be realistic
  - Be verifiable
  - Be uniquely identifiable
  - Not over-constrain the design of the system
Sample questions

- Who is the client?
- Who is the user?
- When do you really need it?
- How much time do we have for this project?
- Will real users be available to help test the product?
- Can we copy or modify something that already exists?
Product questions

- What is the skill level of the users?
- What environment is this product likely to encounter?
- What are the performance and resource constraints?
- What are the safety and security needs?
Meta questions

- Am I asking you too many questions at this time?
- Do my questions seem relevant?
- Are you the right person to answer these questions?
- Are your answers official?
Ending questions

- Is there anything else I should be asking you?
- Is there anything you would like to ask me?
- May I ask you more questions at a later time to help cover things we might have overlooked?
Requirements document (product backlog)

A. Problem
B. Background Information
C. Requirements
   a. Functional requirements
   b. Non-functional requirements
Questions?