Lecture 6: Requirements II, orientation

Lecture goals

- Elicit and analyze requirements from stakeholders
- Get oriented in a new codebase

Requirements analysis

Motivation

- Most projects fail for reasons related to requirements
    - Incomplete
    - Misunderstood
    - Churn
    - Out-of-date
- Therefore, projects would benefit from a more systematic approach to requirements engineering

Requirements steps

- Elicitation & analysis (this lecture)
- Modeling
- Specification (last lecture)

- Heavyweight
    - Document formal specification before beginning design
- Lightweight
    - Relevant requirements developed during sprints
        - But work out system-level requirements upfront
    - Avoid specification unless necessary
        - Models, prototypes clearer to client
    - Sometimes details are important

Stories & scenarios

- Don't start with formal specifications
    - Most clients can't relate to them
    - Difficult to evaluate completeness
- Stories put devs, client on same wavelength
    - Describe actors and their goals

    - High-level, "big picture"
    - Lavish detail about context
        - Helps crystalize alternative viewpoints
        - Refocus by asking which details are relevant
- Scenarios detail interactions with system
    - Agile "user stories" - narrative scenarios with moderate detail
        - Often written on cards
        - Devs break into tasks to estimate effort
        - Prioritized by clients for inclusion in a sprint
        - Postponed stories may be revised with minimal rework
- Structured scenarios provide more detail
    - Tool for clarifying requirements, checking completeness

Usage scenarios

- Illustrates some interaction with a proposed system
- Use specific examples from a user's point of view
- Clarifies many functional requirements
- Especially good for analyzing off-nominal behavior
- Must include:
    - Purpose
    - User or transaction being followed
    - Assumptions about equipment
    - Steps of scenario
- Should consider
    - What could go wrong
    - Concurrent activities
    - Changes to system state
- Avoid system details that pertain to design

Developing scenarios with clients

- Choose a viewpoint
- Identify purpose, actors, equipment, procedure
- Ask clarifying questions

Example: Online exam system

Project goal: Construct a system to allow university students to take exams online from their residences using a web browser.

Viewpoints

- _____
- _____
- _____
- _____

Scenario: typical student

- Purpose: Describe how a typical student uses the system to take an exam
- User: A typical student
  - Describe a particular "typical student"
  - Alice W., a student at Cornell; senior majoring in computer science
  - _____
- Equipment: Her personal laptop, running the Firefox web browser
  - _____
- Steps
  - Alice visits exam system webpage [how does she know which URL to visit?]
  - Alice authenticates with the system [what credentials are required for authentication?]
  - Web page shows list of available exams [is the list personalized for the logged-in user?]
  - Alice selects Exam 1 for CS 1132
  - Web page shows a list of questions. Each one indicates whether it has been answered or not.
  - Alice selects Question 1, which has not yet been answered. [could she have answered questions out-of-order? could she have changed her answer if the question had already been answered?]
  - Question 1 is an essay question. Web page provides an option to type in the answer and an option to upload a file. Alice decides to upload a file. *[what file formats are accepted?]*
  - Later (after answering more questions), Alice selects Question 3, which she previously answered by uploading a file. She deletes that answer and types a new answer directly onto the page. *[could she have viewed/downloaded her previous response? could she have typed an additional answer onto the page without deleting her previous upload?]*
  - Instead of finishing the entire exam in this session, Alice decides to save her progress and resume taking the exam later. She closes her web browser. *[is this always allowed?]*

    - Later, Alice returns to the system, finishes answering all of the questions, and submits her exam. *[what if she submitted without answering every question? is anyone notified when she submits?]*

Scenarios debrief

- Good for eliciting, clarifying functional requirements
- Captures *requirements* for UI, but UI *design* details should be left out
- Complex systems need many scenarios
- Include off-nominal scenarios
    - How can an action be undone? (e.g. financial reversal)
    - How will inconsistencies be handled? (e.g. inventory)
    - Can abuse be detected/prevented? (e.g. submit exam from two browsers)
    - Error handling and recovery (e.g. database loss)

Requirements modeling

Future lecture

Orientation

- Learn to use the product
    - Quick-start guide
- Read project documentation
    - Contributor guide
    - Developer setup
    - Architecture/system design
- Learn to build and test the product
    - Break a test
- Look for landmarks
    - Where does execution start?
    - Search for UI keywords
    - Add tracing statements to confirm hypotheses
- Read tickets/changesets
    - Use “blame” to discover which changeset yielded some code of interest
- Make and observe a change
- Explore code by following methods and types to declarations (requires IDE)
    - Look for commonly reoccurring entities
- Do not try to understand the entire system
    - Accumulate a “working model” of common operations