CIS 422/522

Use Case Summary
In-class Exercise

Problems

- How to convey typical usage scenarios to stakeholders in a way that all can understand
  - Customers, marketers, architects, developers, testers
  - Provide a lightweight means for exploring requirements
- How to express, quickly, key requirements for users in a standardized way
- How to provide a basis for system testing
- How to identify issues for prototyping
- How to start thinking about traceability from requirements to architecture

"Use Cases" can be an effective technique

Use Cases

- Use Case: a story describing how the system and a user interact to accomplish a user task
- A form of *User Centered Analysis* – capturing requirements from the user's point of view
  - Identify capabilities required by different users
  - Solve the right problem
  - Describe the "business logic" of the system
- Use cases specify a *subset of functional requirements*
  - Only system behavior observable to the user
  - Does not typically address quality requirements
- Use cases should not specify design or implementation (including UI design)

Scenario Analysis Process

Applying scenario analysis in the requirements process

- Requirements Elicitation
  - Identify stakeholders who interact with the system
  - Collect "user stories" - how people would interact with the system to perform specific tasks
- Requirements Communication (ConOps)
  - Record as use-cases with standard format
  - Use templates to standardize, drive elicitation
- Requirements verification and validation
  - Review use-cases for consistency, completeness, user acceptance
  - Apply to support prototyping
  - Verify against code (e.g., use-case based testing)

Identifying Actors

- Actors – identifies the roles different users play with respect to the system
  - Roles represent different classes of users (users with different goals)
  - Actors carry out use cases
- Helps identify requirements for different kinds of users
  - "How would depositors use the system?"
  - "How would a library patron use the system?"
- Diverse classes of users may very different goals and require different interfaces
  - E.g., users vs. administrators vs. content providers

Scenario Elicitation

- Each class of actor is interviewed and/or observed
  - How do you do task T?
  - How will the user interact with the system to do X?
- Collect in the form of "user stories"
  - Documented as scenarios (informal or standardized)
  - Identify relative priorities of tasks
  - Resolve conflicts, tradeoffs

Creating Use Cases (Basic)

- Identify a key *actor* and *purpose*
  - The purpose informs the use case title and description
- Identify the main flow (ideal path) from the starting point to the result
  - Preconditions: anything that must be true to initiate the Use Case
  - Trigger: event, if any, initiating the Use Case
  - Basic Flow: sequence of interactions from the trigger event to the result
  - Alternative Flows: identify sequences branching off the Basic Flow
  - Exceptions: identify responses to error conditions

Guidelines for Good Use Cases

- Use Cases should express requirements, not design
  - Focus on import *results* that provide *value* to specific actors
    - I.e., if nobody really cares about the outcome, it is not a good use case
  - Focus on *what* the actor is doing, not the details of *how*
    - Not: "The user left-clicks on the radio button labeled *Balance* and presses the *Enter* button"
    - "The user elects the option to view the balance."
- Looking for a small number of use cases that capture the most important interactions
  - Read the IBM Use Case paper

1 Brief Description

This use case describes how the Bank Customer uses the ATM to withdraw money to his/her bank account.

2 Actors

2.1 Bank Customer
2.2 Bank

3 Preconditions

There is an active network connection to the Bank.
The ATM has cash available.

4 Basic Flow of Events

1. The use case begins when Bank Customer inserts their Bank Card.
2. Use Case: Validate User is performed.
3. The ATM displays the different alternatives that are available on this unit. [See Supporting Requirement SR-xxx for list of alternatives]. In this case the Bank Customer always selects "Withdraw Cash".
4. The ATM prompts for an account. See Supporting Requirement SR-yyy for account types that shall be supported.
5. The Bank Customer selects an account.
6. The ATM prompts for an amount.
7. The Bank Customer enters an amount.
8. Card ID, PIN, amount and account is sent to Bank as a transaction. The Bank Consortium replies with a go/no go reply telling if the transaction is ok.
9. Then money is dispensed.
10. The Bank Card is returned.
11. The receipt is printed.

5 Alternative Flows

5.2 Wrong account

If in step 8 of the basic flow the account selected by the Bank Customer is not associated with this bank card, then

1. The ATM shall display the message "Invalid Account – please try again".

2. The use case resumes at step 4.|

9

Example Use Case

- Avoids design decisions
- References other use cases
- References more precise definitions where necessary
- Some terms need further definition (e.g. PIN)

Questions?

Deliverables Walkthrough

- Consider: What kinds of questions should your documents answer?
  - Assume a manager unfamiliar with the project is reviewing your status
  - Would your documents answer key questions about the project goals and current status?
- *Team page*: Who is on the team and what are their skills?
- *Project plan*
  - Who is responsible for which tasks?
  - What are the anticipated risks and what are you doing to mitigate them?
  - What is your development process and how does it help address the risks?
  - Detailed Schedule & Milestones
    - What is the project schedule of tasks and deliverables?
    - What is the current status relative to schedule?

Walkthrough (2)

- *Software Requirements*
  - 2. ConOps: What capabilities will the software provide the user or customer?
  - 3. Behavioral Requirements: What are the detailed technical requirements?
    - Specific inputs accepted & outputs generated
    - Detailed behavior of any computation (e.g., sort, error responses)
  - 4. Quality Requirements: objective requirements for software qualities (e.g., reliability, performance)
- *Software Design*
  - Architecture: How is the software organized into components? How does it work (function)? Where is each requirement implemented (traceability)?
  - Module Interfaces: What are the component interfaces?

Walkthrough (3)

- *Quality Assurance*: How will you check whether the software satisfies functional and quality requirements?
  - Reviews: Which artifacts/properties will be checked by review?
  - Test Plans: How will you test the software?
- *User Documentation*: How will users understand how to install and use the application?
- *Code Documentation:* What do I need to know to find parts of the code responsible for implementing any given requirement or part of the design?
  - How is the code organized in the repository?
  - What does this code component do?