Lecture 5: Requirements Engineering II

COSI 120b, Principles of Software Engineering
Your Requirements

- Customer
- UI Designer
- Tester
- Sales
- End User
Your Requirements

- What did they look like?
- How specific were they?
- Were they right?
- Were they possible?
- Could they be integrated with everyone’s requirements into the larger requirements set?
Your Requirements

• The stakeholders may not (probably do not) have a view of the complete system
  – Myopic views lead to myopic requirements

• The stakeholders do not know all requirements of the system
  – Does the UI guy know what testing parameters the QA guy wants?

• Multiple stakeholders are needed
Your Requirements

• So how do we integrate disparate and conflicting requirements into a single requirements document?
• How do we handle conflicts that must be rejected?
  – Infeasible and impossible requirements
• How do we ensure that we are successful?
Overview

• Agreement
• Building Requirements
• Traceability
• Elicitation Techniques
• Conclusions
• What’s Next?
• Assignment 1b
Agreement

- Collaboration, not conflict
- Strategy
- Commitment
- Know who the stakeholders are
  - Know their level of “importance”
- Set the goal posts
  - Do not leave without a metric for success
Agreement

• Validation
  – Always validate the formal requirements
  – Unvalidated requirements are not complete requirements
  – Validate each requirement
Validation Questions

• For each requirement:
  – Consistent with project objectives
  – Level of abstraction / technical details
  – Necessity
  – Bounded and unambiguous
  – Attribution
  – Conflicting requirements
  – Testability

• For the requirements model:
  – Information, function, and behavior
  – Partitioning
  – Patterns and validation of patterns
Building Requirements

- So the stakeholders have told us what they want
- And we have negotiated these requirements into a unified set of informal requirements
- What do we do now?
Building Requirements

• Formalization of requirements
  – Remove ambiguity
    • The developer may interpret an ambiguous requirement incorrectly
    • The requirement may be written down too vaguely during elicitation
  – Understand the agreed to requirements
    • What a requirement actually entails
  – Remove redundant requirements
Building Requirements

• How do we formalize?
  – An agreed to, unambiguous method
    • Any method that the team agrees to is acceptable
    • However, there are some well-known methods
      – UML
  – What do we want to capture?
    • Use cases
      – A description of the system interaction
Use Cases

• A use case is a series of steps of the form:
  – User A does x
  – System responds y
  – User A does x’
  – System responds y’

• Describe the use case of taking money out of your checking account via an ATM
Use Cases

• How do you formalize the use case?
  – UML
Use Cases

• Template (Cockburn, 2001)

Use-case
Primary actor
Goal in context
Preconditions:
Trigger:
Scenario:
Exceptions:
Priority
Availability
Frequency
Channel to actor
Secondary actors
Channels to secondary actors
Open issues
Use Cases

- Template for the ATM
Analysis Models

• Detailing the use cases with system information
  – The analysis model is the start of the design process

• Four parts to the analysis model
  – Scenario elements
  – Class-based elements
  – Behavioral elements
  – Flow elements
Analysis Models

• Scenario-based elements
  – UML, templates

• Class-based elements
  – What are the system objects?

• Behavioral elements
  – State diagrams
  – System states
    • i.e. once the ATM has your PIN verified there are new actions available

• Flow elements
  – Transformation of information
Analysis Patterns

• What is a pattern?
  – A well-known solution to a common problem
    • Originated by C. Alexander, an architect, for constructing buildings
  – We will talk about patterns throughout this class
    • Design Patterns
    • Refactoring Patterns
    • Analysis Patterns
  – Why are patterns important?
Analysis Patterns

• Common analysis models to requirements that are commonly found in a given domain
  – i.e. no matter what your ATM looks like, certain behaviors are always needed

• By leveraging a pattern, you get:
  – The experience behind the pattern
  – A reasonable expectation of other developers understanding your intent and usage of the pattern
Traceability

• Source traceability
  – Who are the stakeholder(s)?

• Dependency traceability
  – Which requirements are dependent on which other requirements?

• Subsystem traceability
  – Identification of subsystems and their requirements

• Interface traceability
  – Identification of the system interfaces (both developer APIs and Uis) and which requirements govern them
Elicitation Techniques

• Collaboration
  – Negotiation, not conflict
  – Meeting artifacts
    • Whiteboards, flip charts, email, etc
  – Agenda

• Elicitation work products
  – Statements of need and feasibility
  – Bounded scope
  – Stakeholders
  – Technical environment
  – Usage scenarios
  – Prototyping needs
  – List of requirements
Conclusions

• How do good requirements stave off complexity?
  – Certainty of the task
  – Clear milestones

• How does a good requirements process stave off complexity?
  – Understanding of the changing task
  – Control of how the task changes
Conclusions

• What tools do you now have?
  – Understanding the many faces of the stakeholders
  – Understanding of what their priorities are
  – Understanding how to handle sub-optimal requirements
    • Underspecified
    • Impossible
    • Conflicting
What’s Next?

• The term project
  – Put together groups of 3 - 4 people by next class
  – Think of a big application you want to put together
  – Email me your groups and 1 paragraph on your application by next class

• Developer Collaboration
Assignment 1b

• Tonight, I will send out sets of requirements
• Come up with:
  – Formal requirements, where possible. Use any representation you think is appropriate, including
    • Use cases that satisfy these requirements
    • UML diagrams
    • Templates
  – Where you can’t do formal requirements, explain why.
• Explain how you came up with the set
• Explain what you would do if you got all the stakeholders in a room together
• Due 2/13/2005 by 5pm in my mailbox (seth@cs.brandeis.edu)
• Turn in the UML or other non-electronic work in class on 2/14/2005