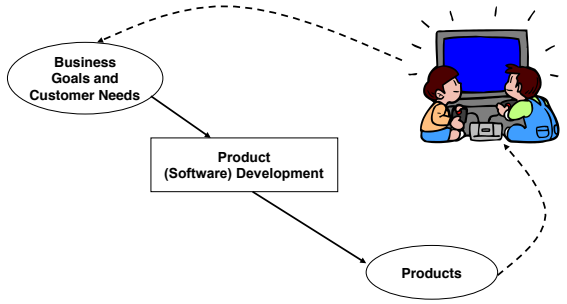
CIS 422/522

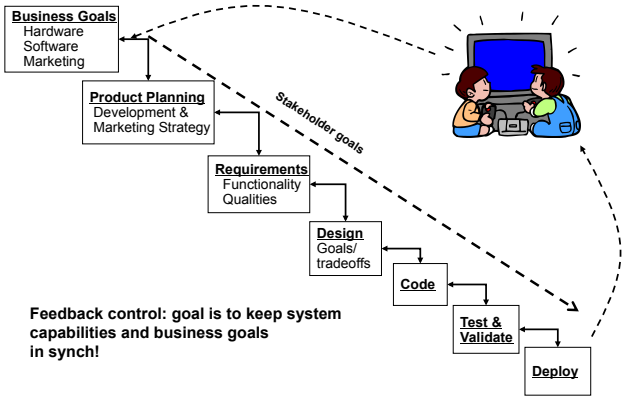
Software Requirements
& a Little Quality Assurance

10,000 ft. View



What should the development process accomplish?

Product Development Cycle



Feedback control: goal is to keep system capabilities and business goals in synch!

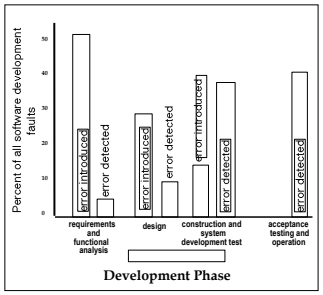
What is a “software requirement?”

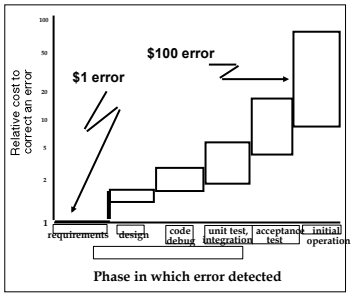
- *Definition*: A description of something the software must do or property it must have
- The set of system requirements denote the problem to be solved and any constraints on the solution
  - Ideally, requirements specify precisely what the software must do without describing how to do it
  - Any system that meets requirements should be an acceptable implementation

Importance of Getting Requirements Right

1. The majority of software errors are introduced early in software development

2. The later that software errors are detected, the more costly they are to correct





Requirements Phase Goals

- What does "getting the requirements right" mean in the systems development context?
- Only three goals
  1. Understand precisely what is required of the software
  2. Communicate that understanding to all of the parties involved in the development (stakeholders)
  3. Control production to ensure the final system satisfies the requirements
- Sounds easy but hard to do in practice
- Understanding what makes these goals difficult to accomplish helps us understand how to mitigate the risks

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements...No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later."

F.P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering"

What makes requirements difficult?

- Comprehension (understanding)
  - People don't (really) know what they want (…until they see it)
  - Superficial grasp is insufficient to build correct software
- Communication
  - People work best with regular structures, conceptual coherence, and visualization
  - Software's conceptual structures are complex, arbitrary, and difficult to visualize
- Control (predictability, manageability)
  - Difficult to predict which requirements will be hard to meet
  - Requirements change all the time
  - Together can make planning unreliable, cost and schedule unpredictable
- Inseparable Concerns
  - Many requirements issues cannot be cleanly separated (I.e., decisions about one necessarily impact another)
  - Difficult to apply "divide and conquer"
  - Must make tradeoffs where requirements conflict

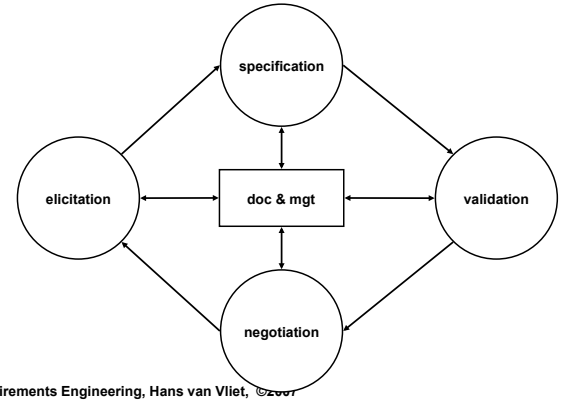
Requirements Process

Understand, Communicate & Control

Managing requirements difficulties requires having a good process

1. Requirements Understanding (Understand)
   1. Elicitation - How do we establish "what people want?"
   2. Negotiation - How do we resolve stakeholder conflicts?
2. Requirements Specification (Communicate)
   1. Concept of Operations (ConOps) - How do we communicate with non-programmer audiences?
   2. Software Requirements Specification (SRS)- How do we specify precisely what the software must do?
3. Requirements Validation and Verification (Control)
   - How do we establish that we have the right requirements?
   - How do we establish that the implementation meets the requirements?

RE Process from Text: Same Idea



SE, Requirements Engineering, Hans van Vliet, ©2007

3. Validation and Verification

- Part of *Quality Assurance* – provides feedback in the feedback-control-loop
- *Validation*: activities to answer the question – "Are we building a system the customer wants?"
- *Verification*: activities to answer the question – "Are we building the system consistent with all specifications?"
- Most familiar verification activity is functional testing

Project V&V

- QA Goal: How can we establish whether the development is under control?
- Project sub-questions:
  - How will you establish that the system does what it should?
  - What is the role of testing?
  - What can testing establish about system quality (and what can't it)?
  - How will you write test cases?
    - E.g., for the voting system?

Understand, Communicate & Control

Managing requirements difficulties requires having a good process

1. Requirements Understanding (Understand)
   1. Elicitation - How do we establish "what people want?"
   2. Negotiation - How do we resolve stakeholder conflicts?
2. Requirements Specification (Communicate)
   1. Concept of Operations (ConOps) - How do we communicate with non-programmer audiences?
   2. Software Requirements Specification (SRS)- How do we specify precisely what the software must do?
3. Requirements Validation and Verification (Control)
   - How do we establish that we have the right requirements?
   - How do we establish that the implementation meets the requirements?

1.1 Elicitation

- Goal: Understand precisely what is required of the software
  - Answer the question, "What do the stakeholders want?"
  - Stakeholder: define as anyone with a valid interest in the outcome of a software development
- Inherently open-ended, ambiguous question
- Addressed by a number of elicitation methods
  - Interview – traditional standard
  - Focus groups
  - Prototyping
  - Scenario analysis (next), etc.
- All have differing costs, strengths, and weaknesses. None is a complete solution
  - Use more than one approach
  - Check the results *early and often*

1.2 Requirements Negotiation

- or "Why the customer is not always right."
- Stakeholders' requirements often conflict
  - Needs of different customers/users may conflict
    - E.g., Salesmen want convenience and speed, management wants security and accountability
  - Developer's needs may conflict with customer's
    - E.g., development cost vs. customer desires
- Choosing which requirements should be addressed and their relative importance requires *negotiation and tradeoffs* among stakeholders

2. Requirements Specification

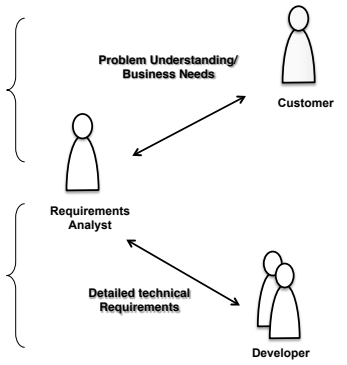
- Goal: Communicate requirements understanding to all system stakeholders
- Q: What kinds of information need to be communicated?
  - System context
    - System stakeholders
    - Business goals
    - System purpose
    - Interfacing systems (if any)
  - System requirements
    - Behavioral requirements
    - Quality requirements

Purposes and Stakeholders

- Many potential stakeholders using requirements for different purposes
  - Customers: document what should be delivered, may provide the contractual basis for the development
  - Managers: provides a basis for scheduling and a yardstick for measuring progress
  - Software Designers: provides the "design-to" specification
  - Coders: defines the range of acceptable implementations and is the final authority on the outputs that must be produced
  - Quality Assurance: basis for validation, test planning, and verification
  - Also: potentially Marketing, regulatory agencies, etc.

Needs of Different Audiences

- Customer/User
  - Focus on problem understanding
  - Use language of problem domain
  - Technical if problem space is technical
- Development organization
  - Focus on system/software solutions
  - Use language of solution space (software)
  - Precise and detailed enough to write code, test cases, etc.



Two Kinds of Requirements Documentation

- Communicate with stakeholders who understand the problem domain but not necessarily programming :
  - e.g. customers, users, marketing
  - Do not understand computer languages but may understand technical domain-specific languages
  - Must develop understanding in common languages
  - Role of ConOps (Concept of Operations)
- Communicate with developers: sufficiently precise and detailed to code-to, test-to, etc.
  - Stated in the developer's terminology
  - Addresses properties like completeness, consistency, precision, lack of ambiguity
  - Role of SRS (Software Requirements Specification)
- For businesses, these may be two separate documents

SRS Template

1. Introduction

1.1 Intended Audience and Purpose

<Describes the set of stakeholders and what each stakeholder is expected to use the document for. If some stakeholders are more important than others, describes the priorities.>

1.2 How to use the document

<Describes the document organization. This section should answer for the reader: "Where do I find particular information about X?">

2. Concept of Operations

<Use this section to give a detailed description of the system requirements from a user's point of view. The ConOps should be readable by any audience familiar with the application domain but not necessarily with software. The ConOps should make clear the context of the software and the capabilities the system will provide the user.>

Informal, user centric

2.1 System Context

<Specify the system boundaries including, particularly, the inputs and outputs. May include an illustration or context diagram. >

2.2 System capabilities

<System capabilities may be described in prose or with informal scenarios.>

3. Behavioral Requirements

<Specification of the observable system behavior.>

3.1 System Inputs and Outputs

3.2 Detailed Output Behavior

<A black box specification of the visible, required behavior of the system outputs as a function of the system inputs. Tables, functions, use cases or other methods of specification may be used.>

Formal, technical

Documentation Approaches

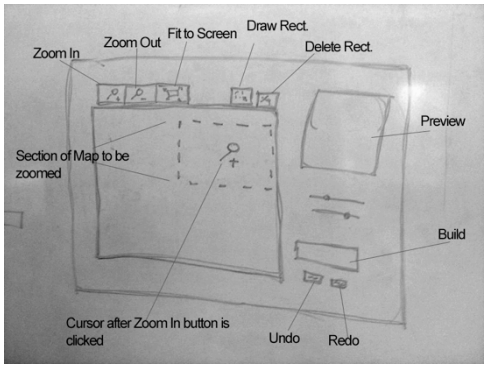
- Informal requirements to describe the system's capabilities from the customer/user point of view
  - Purpose is to answer the questions, "What is the system for?" and "How will the user use it?"
  - Tells a story: "What does this system do for me?"
  - Focus on communication over rigor
- More formal, technical requirements for development team (architect, coders, testers, etc.)
  - Purpose is to answer specific technical questions about the requirements quickly and precisely
    - "What should the system output for this set of inputs?"
    - Reference, not a narrative, does not "tell a story"
  - Goal is to develop requirements that are precise, unambiguous, complete, and consistent
  - Focus on precision and rigor

Informal Specification Techniques

- Most requirements specification methods are informal
  - Natural language specification
  - Use cases
  - Mock-ups (pictures)
  - Story boards
- Benefits
  - Requires little technical expertise to read/write
  - Useful for communicating with a broad audience
  - Useful for capturing intent (e.g., how does the planned system address customer needs, business goals?)
- Drawbacks
  - Inherently ambiguous, imprecise
  - Cannot effectively establish completeness, consistency
- However, can add rigor with standards, templates, etc.

Mock-up Example

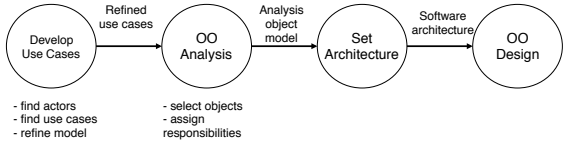
Can use simple technology



Analysis and Informal Specification with Use Cases

Use Cases

- Often done as a prelude to OO modeling
- A form of *User Centered Analysis* – capturing requirements from the user's point of view
  - Goal of helping identify user needs
  - Solve the right problem
- Best fit with operational requirements (mission statement, ConOps, BRD, etc.)



Scenario Analysis Process

Applying scenario analysis in the requirements process

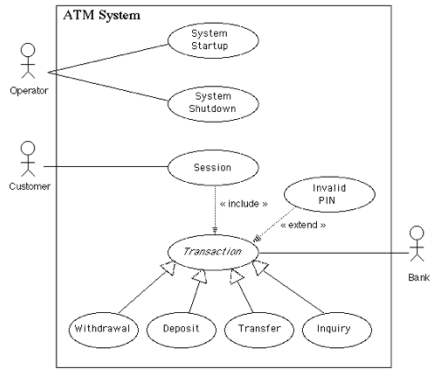
- Requirements Elicitation
  - Identify stakeholders who interact with the system
  - Collect "user stories" - how people would interact with the system to perform specific tasks
- Requirements Specification
  - Record as use-cases with standard format
  - Use templates to standardize, drive elicitation
- 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 a role different users plays with respect to the system
  - Roles represent different classes of users (use the system with different goals)
  - Actors carry out use cases
- Primarily useful in identifying different kinds of use cases
  - "How would depositors use the system?"
  - "How would a library patron use the system?"
- Important to keep in mind that there may be several diverse classes of users with very different goals and interfaces
  - E.g., users vs. administrators vs. content providers, etc.

UML Graphic Example

http://www.math-cs.gordon.edu/local/courses/cs211/ATMExample/



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

Simple Story and Scenarios

As an Account Holder
I want to withdraw cash from an ATM
So that I can get money when the bank is closed

Scenario 1: Account has sufficient funds
Given the account balance is \$100
And the card is valid
And the machine contains enough money
When the Account Holder requests \$20
Then the ATM should dispense \$20
And the account balance should be \$80
And the card should be returned

Scenario 2: Account has insufficient funds
Given the account balance is \$10
And the card is valid
And the machine contains enough money
When the Account Holder requests \$20
Then the ATM should not dispense any money
And the ATM should say there are insufficient funds
And the account balance should be \$20
And the card should be returned

Terminology

- Scenario – description of a sequence of interactions between a user and the system from the user's point of view
  - What does the user see or do
  - What does the system do in response
- Use Case – a set of scenarios related by a common user *goal*
  - Goal – an objective the user is employing the system to achieve
  - Scenarios represent different possible outcomes (nominal case, error case, etc.)

Use Cases

1 Use Case: Manage Reports

1.1 Description

This Use Case describes operation for Creating, Saving, Deleting, Printing, Exiting and Displaying reports.

1.2 Actors

User
Project database

1.3 Triggers

Program Manager selects operations from menu.

1.4 Flow of events

1.4.1 Basic Flow

1. **User chooses desired report by selecting "Report" -> "Open" from the menu bar**
2. **System displays report to screen**
3. **User selects desired report layout using Use Case Specify Report**
4. **Steps 2 and 3 are repeated until user is satisfied**
5. **User can Save or Print report using use case Save Report or Print Report**
6. **User Exits report by selecting "Exit" from the "File" menu**

1.4.2 Alternative Flows

1.4.2.1 Create New Report

1. **User selects "Create New Report" from file menu**
2. **…**

1.4.2.2 Delete Report

……

1.4.3 Preconditions

etc

A systematic approach to use cases

- **Uses a standard template**
- **Easier to check, read**
- **Still informal**

Use Case Contents (Generic)

- Use case identifier
- Summary – summary of use case
- Actors – roles enacting use case
- Scenarios
  - Basic scenario – the normal case
  - Alternative scenarios – other ways to reach goal
  - Exceptions – problem scenarios
- Trigger – what causes the use case to start
- Assumptions
- Preconditions – what must be true before the interaction can occur
- Post conditions – what must be true after the interaction occurs

Basic Scenario

- Sequence when the user proceeds to his goal as system designers intend

1.Customer puts card into ATM card slot and enters PIN number.
2.Card verified and main menu presented.
3.Customer selects the transaction services menu and the corresponding menu is displayed.
4.Customer selects ``automated payment service'' and is prompted for the recipient's account number.
5.Customer enters recipient's account number.
6.Account verified and a menu with payment schedules is presented.
7……..

Exceptions

- What is the scenario if the customer enters the wrong PIN?

1. Customer puts card into ATM card slot and enters PIN number.
2. Incorrect PIN identified and error message "Incorrect PIN" and error menu displayed.
3. Customer selects the "try again" option
4……..

Assignment

- For Monday: all team members participate
  - First draft of ConOps
  - First set of test cases
- Apply Use Cases
  - Answer the questions, “What does the system do?” and “How will the user use it?”
  - Should have use cases for all major system interactions
- Use to help drive customer interaction (elicitation)
  - Derive questions for customer (me)?
- Use to establish initial set of test cases

Summary

- Requirements characterize “correct” system behavior
- Being in control of development requires:
  - Getting the right requirements
  - Communicating them to the stakeholders
  - Using them to guide development
- Requirements activities must be incorporated in the project plan
  - Requirements baseline
  - Requirements change management

Questions?

Product Development Cycle