CIS 422/522

Deliverables Walkthrough
Software Requirements
Use Cases

I'LL DESIGN THE SYSTEM AS SOON AS YOU GIVE ME THE USER REQUIREMENTS.
BETTER YET, YOU COULD BUILD THE SYSTEM, THEN I'LL TELL YOUR BOSS THAT IT DOESN'T MEET MY NEEDS.
I DON'T MEAN TO FRIGHTEN YOU, BUT YOU'LL HAVE TO DO SOME ACTUAL WORK.
THAT'S CRAZY TALK.
www.dilbert.com

Today

- Change to the usual lecture format (flipped classroom experiment)
  - Brief lecture on some basic issues in Requirements
  - Description of Use cases (a specific approach you will use)
  - Hands-on exercise defining Use Cases
- Full lecture will be recorded and posted as soon as possible
  - Contains material covered in tests so you need to watch it

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?
- *Project plan*
  - Who is responsible for which tasks?
  - What are the anticipated risks and what are you doing about 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?

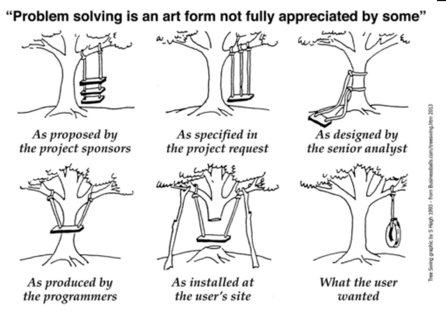
Walkthrough (2)

- *Software Requirements*
  - ConOps: What capabilities will the software provide the user or customer?
  - Behavioral Requirements: What are the detailed technical requirements?
    - Specific inputs accepted & outputs generated
    - Detailed behavior of any computation (e.g., sort, error responses)
  - Quality Requirements: objective requirements for software qualities (e.g., reliability, performance)
- *Software Design*
  - Architecture: How is the software organized into components? Important relationships between components?
  - Module Interfaces: What are the component interfaces?

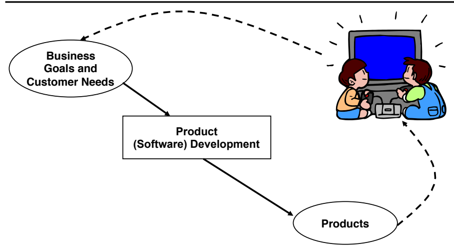
Walkthrough (3)

- *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?

Understanding Software Requirements (and why we get it wrong so often)

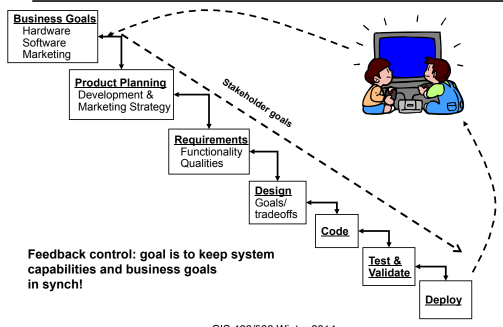

"Problem solving is an art form not fully appreciated by some"
As proposed by the project sponsors
As specified in the project request
As designed by the senior analyst
As produced by the programmers
As installed at the user's site
What the user wanted
Tree Swing graphic by S Hogh 1993 - from Businessballs.com/treeswing.htm 2013

10,000 ft. View


Business Goals and Customer Needs
Product (Software) Development
Products

What should the development process accomplish?

Product Development Cycle



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

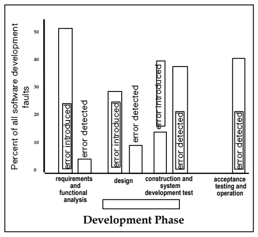
What is a "software requirement?"

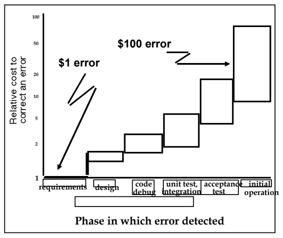
- *Definition*: A description of something the software must do or property it must have
- The set of system requirements denote the problem to be solved and any constraints on the solution
  - Ideally, requirements specify precisely what the software must do without describing how to do it
  - Any system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

Implications

- Do not assume that you know what the customer wants
- Do not assume that the customer knows exactly what he/she wants
  - At least until they see it
- The hard part: arriving at a consistent, mutual understanding of what should be built sufficiently detailed to code to
- Our "customer interactions" focus on getting past assumptions to precise answers
  - Simulates customer who understands domain but not programming
  - Does not answer questions that are not asked
  - Answers the question asked, not the question you should have asked

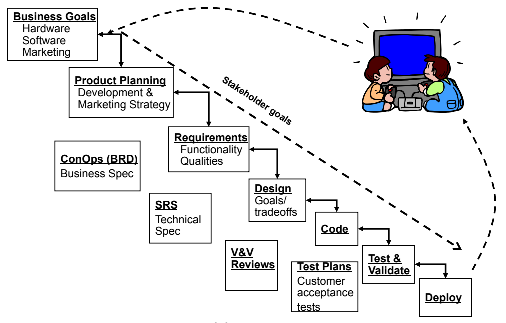
Requirements Process

Understand, Communicate & Control

A good process helps manage requirements difficulties requires having

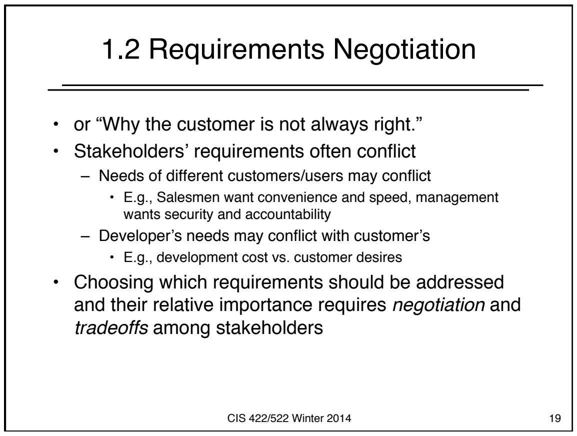
1. Requirements Understanding (Understand)
   – Elicitation - How do we establish “what people want?”
   – Negotiation - How do we resolve stakeholder conflicts?
2. Requirements Specification (Communicate)
   – Concept of Operations (ConOps) - How do we communicate with non-programmer audiences?
   –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

Related Products



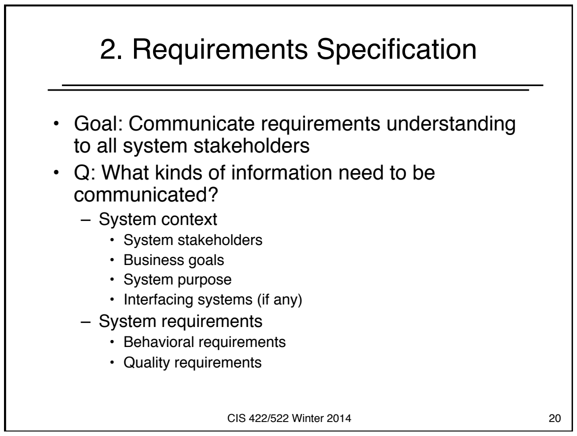
1.1 Elicitation

- Goal: Understand precisely what is required of the software
  – Answer the question, “What do the stakeholders want?”
  – Stakeholder: 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*



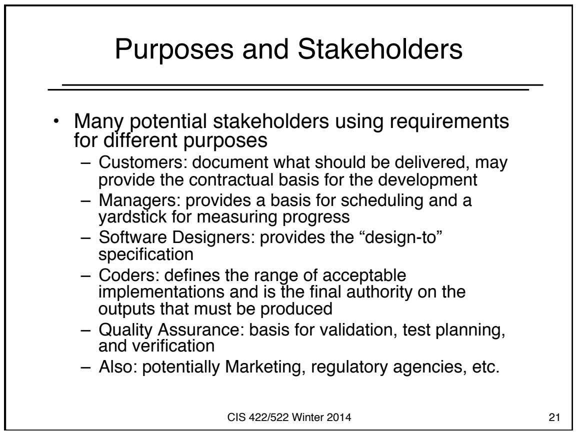
1.2 Requirements Negotiation

- 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

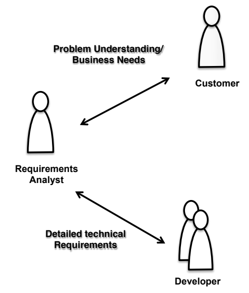


Purposes and Stakeholders

- 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.


Problem Understanding/ Business Needs
Customer
Requirements Analyst
Detailed technical Requirements
Developer

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?">

Informal, user centric

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.>

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.>

Formal, technical

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.>

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.

Analysis and Informal Specification with Use Cases

Problems (1)

- How to convey typical usage scenarios to stakeholders in a way that all can understand
  - Customers
  - Architects
  - Developers
  - Testers
  - Marketers
- How to express, quickly, key requirements for users in a standardized way
  - Provide a lightweight means for exploring requirements

Problems (2)

- 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
  - Goal of helping identify user needs
  - 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

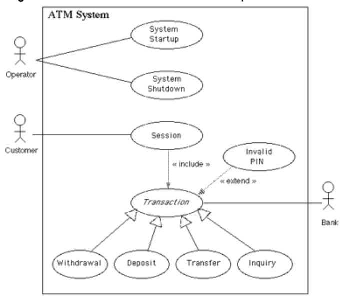
- Requirements Elicitation
  - Identify stakeholders who interact with the system
  - Collect “user stories” - how people would interact with the system to perform specific tasks
- Requirements Communication (ConOps)
  - Record as use-cases with standard format
  - Use templates to standardize, drive 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

UML Graphic Example

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



Scenario Elicitation

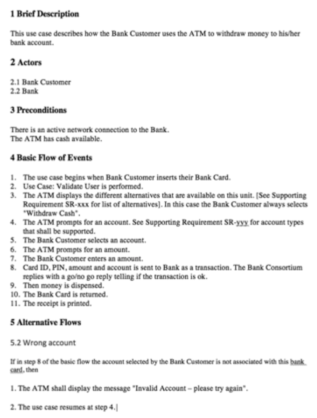
- Each class of actor is interviewed and/or observed
  - How do you do task T?
  - How will the user interact with the system to do X?
- Collect in the form of "user stories"
  - Documented as scenarios (informal or standardized)
  - Identify relative priorities of tasks
  - Resolve conflicts, tradeoffs

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.

Example Use Case

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

37

Use Cases and Prototypes

- Map from Use Cases to Prototypes and vice-versa
  – If it's not clear what to do, or how to do some step in the use case, create a prototype
    - What privileges should a user have?
    - What happens if the user enters an inappropriate value in field *F*?
  – When a prototype clarifies an issue, the use case may need to be revised
    - Three different types of users, with different privileges, must be able to use the system

CIS 422/522 Winter 2014 38

Use Cases and Design

- Each step in a use case will result in a call on an access program for a module, e.g., an access method on the public interface of a class
  – When design is complete, it should be possible to trace from the use case to the modular design
  – One step in the chain of traceability from requirements to implementation and testing

CIS 422/522 Winter 2014 39

Use Cases and Testing

- Map from Use Cases to System Tests
  - Use cases are representative of common user scenarios (operational scenarios)
  - Use cases provide a start on what's needed for test cases
    - Preconditions
    - Inputs
    - Outputs
    - Exceptional Cases
    - Post-conditions

Address Book Example

- Who are the actors?
- What are the major tasks?
- What are the outcomes?
- What would be an alternative flow?

Summary

- Use cases help elicit requirements
- Use cases help clarify requirements
- Use cases help make it clear what the most useful requirements are
- Use cases help determine system test cases
- Use cases provide a common base of understanding of what interaction the system has with its users, both human and automated
- Use cases are not a precise, unambiguous statement of requirements
- Use cases do not cover the details of every possible interaction of users with the system
- Use cases provide a step on the path of tracing from requirements to design

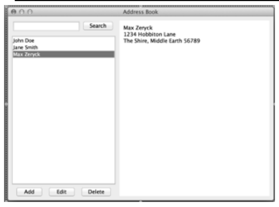
3. Validation and Verification

- 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?”
  - Familiar activity: customer review of prototype
- *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 review?
    - What is the role of testing?
  - Why do we want a QA plan?
    - E.g., for the class project

Customer Validation “Is this OK?”



Max Zeryck
1234 Hobbiton Lane
The Shire, Middle Earth 56789

- Usually get prototypes with questions of the form “Is this OK?”
- What answer are you likely to get?
  - What does the question actually mean?
- What do you actually want from the review?
  - What question do you actually want the customer/reviewer to answer?

A *Proactive* Review is Better

- Think ahead about what kind of questions you want the review to answer
  - Usability of interface?
  - Look and feel?
  - Completeness?
- Then, design the review to answer them
  - Put questions in writing
  - Organize to systematically cover topics
    - E.g., using work-flow or Use 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?