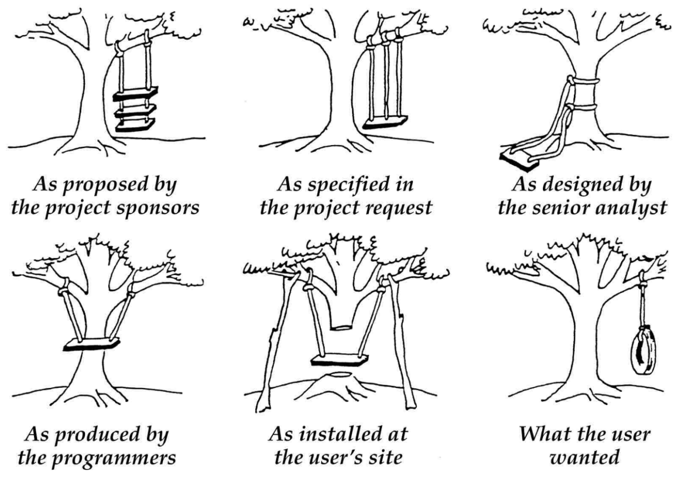
CIS 422/522

Software Requirements 1

Stuart Faulk
Computer and Information Science

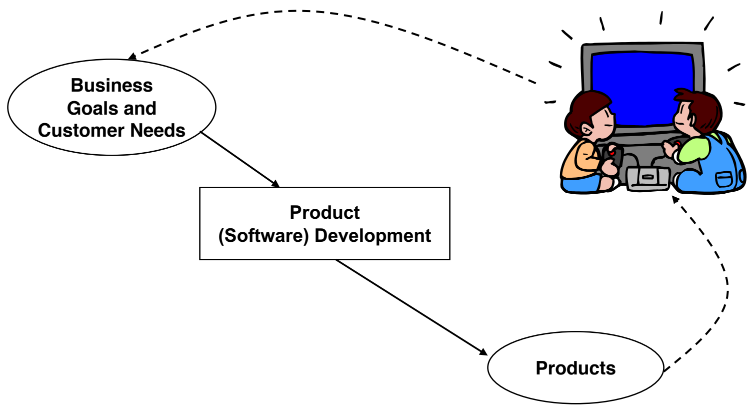
Understanding Software Requirements (and why we get it wrong so often)

"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

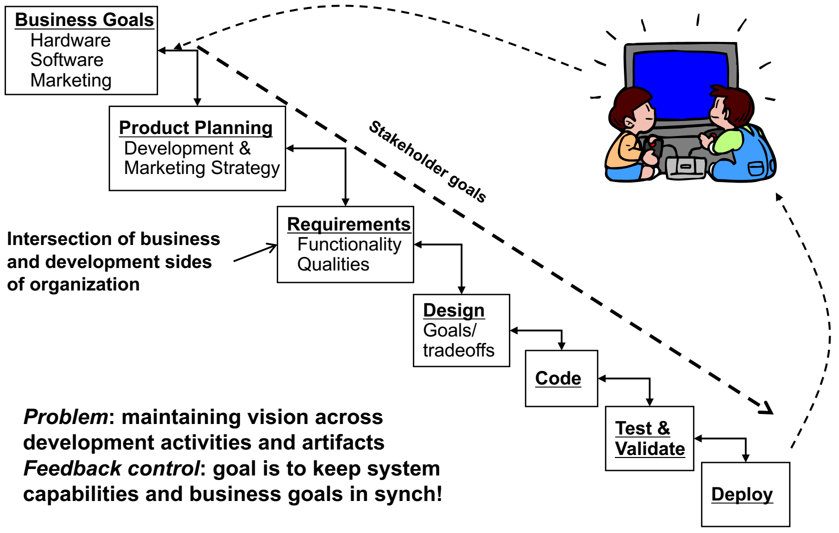
Tree Swing graphic by S Hagh 1993 - from Businessballs.com/treeswing.htm 2013

10,000 ft. View


Business Goals and Customer Needs
Product (Software) Development
Products

What should the development process accomplish?

Product Development Cycle



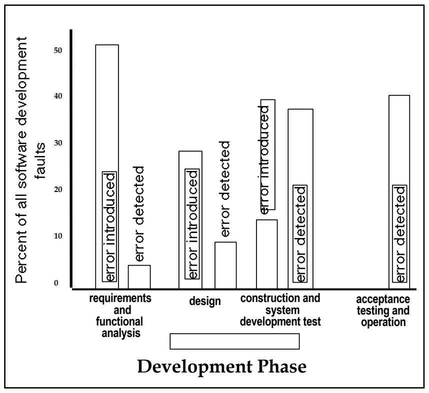
***Problem*: maintaining vision across development activities and artifacts**
***Feedback control*: goal is to keep system capabilities and business goals in synch!**

What is a “software requirement?”

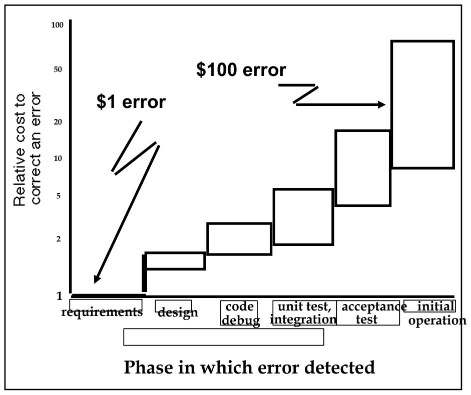
- *Definition*: A description of something the software must do or property it must have
- The set of system requirements denote the problem to be solved and any constraints on the solution
  - Ideally, requirements specify precisely what the software must do without describing how to do it
  - Any system 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 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
- All three goals are inherently difficult
- Must be managed as risks

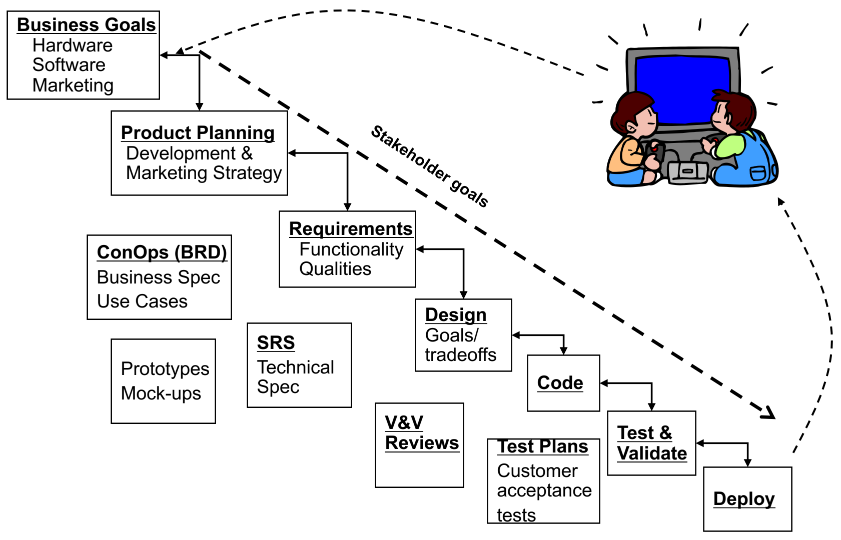
Requirements Process

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
  - Use cases
- 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

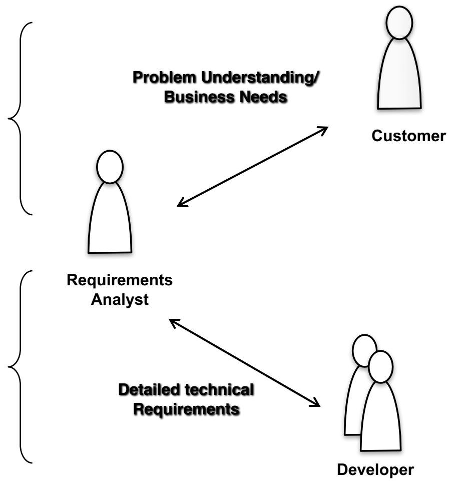
- Goal: Communicate requirements understanding to all system stakeholders
- Q: What kinds of information need to be communicated?
  - System context (link to business objectives)
    - System stakeholders
    - Product business goals
    - System purpose
    - Interfacing systems (if any)
  - System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

SRS Purposes and Stakeholders

- Sits at business/development intersection
- Many potential stakeholders using requirements for different purposes
  - Customers: document what should be delivered
  - Marketing: capabilities to be delivered
  - 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

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
  - Must develop understanding in common language
  - Role of ConOps (Concept of Operations)
- Communicate with developers
  - Stated in the developer's terminology
  - Sufficiently precise and detailed to code-to, test-to, etc.
  - 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

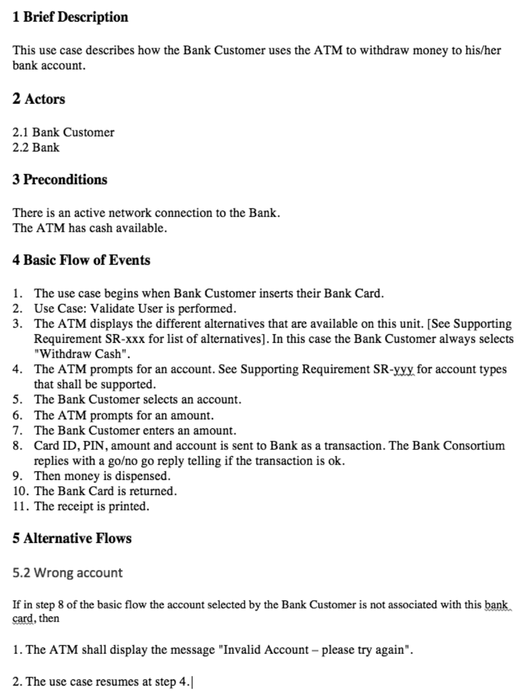
- Informal requirements to describe the system's capabilities from the customer/user point of view
  - Purpose is to answer the questions, "What is the system for?" and "How will the user use it?"
  - Tells a story: "What does this system do for me?"
  - Focus on communication over rigor
- More formal, 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 (may use formal languages)
  - We will only do a little of this

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.

Example: Use Cases

- Informal specification requirements in terms of system capabilities provided to a user
- Each Use Case describes how the system and a user interact to accomplish a user task
  - Specifies (only) functional behavior
  - Captures the "business logic" of the application
- Inherently ambiguous, incomplete
  - Can add rigor with consistent templates, good process, reviews



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)

Supports Requirements Goals

Applying Use Cases in the requirements process

- Requirements Elicitation
  - Identify which capabilities the system should provide to each class of users
  - Collect in terms of problem domain goals
  - Provides basis for prototypes, mockups
- Requirements Communication (ConOps)
  - Record as use-cases with standard format
  - Easy to read and understand for wide audience
- Requirements verification and validation
  - Review use-cases for consistency, completeness, user acceptance
  - Create test cases consistent with use cases
  - Verify against code (e.g., use-case based testing)

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
  - Elicitation and validation activities
  - Specification activities
  - Verification and validation activities
  - Requirements change management

End