Requirements

CITS1220 Software Engineering
Lecture Overview

- What are requirements?
- Five techniques for requirements gathering
- Testing requirements
- Requirements Pitfalls
What are requirements?
Stakeholders for SW systems

- Customers or Clients
  - Those who pay for the software
- Users
  - Those who use the software
- Software developers
- Development Managers
- External agencies: regulators, local law, competitors
Defining the Problem

- A problem statement can be expressed as:
  - A **difficulty** the users or customers are facing such as adequate support for making complex decisions
  - Or as an **opportunity** that will result in some benefit such as improved productivity or sales.

- A good problem statement is short and succinct
Defining the Scope

- Narrow the **scope** by defining a more precise problem
  - List all the things you might imagine the system doing
    - Exclude some of these things if scope is too broad
    - Determine high-level goals if scope is too narrow

- Example: A university registration system

Initial list of problems with very broad scope:
- browsing courses
- room allocation
- registering
- exam scheduling
- fee payment

Narrowed scope:
- browsing courses
- room allocation
- registering
- exam scheduling
- fee payment

Scope of another system:
- room allocation
- exam scheduling
- fee payment
What is a Requirement?

- It is a *statement* describing either
  - 1) an aspect of what the proposed system must do, or
  - 2) a constraint on the system’s development.
- In either case it must contribute in some way towards solving the customer’s problem;
- The set of requirements as a whole represents a negotiated agreement among the stakeholders
- A collection of requirements is a *requirements document.*
Business vs System Reqs

- Experienced systems analysts distinguish between specifying
  - what a proposed new system will do and specifying
  - what the user organization will be able to do given a solution to its problem. That solution will often but not always be a new system.

- BR often expressed as lists of discrete goals or objectives that will, when quantified, serve as justification for the project.

- AKA engineering reqs if not for a business app
Business Requirements (cont)

- Some Examples
  - To reduce warehouse inventory cost by 30%.
  - To ship 95% of orders within 12 hours of receipt.
  - To eliminate fines for late filing of OSHA compliance reports.

- In theory, we can consider ways of satisfying them **that do not call for a new system.**

- Unfortunately, Business Requirements is the most often overlooked phase in the life cycle.
Types of **System** Requirements

- **Functional** requirements
  - Describe *what* the system should do;
  - A service the system provides

- **Quality** requirements
  - *Constraints* on the design such as:
    - Reliability, Speed, Size, Efficiency, Portability,

- **Platform** requirements
  - *Constraints* on environment and technology

- **Process** requirements
  - *Constraints* on the project plan and development methods
Functional Requirements

- What *inputs* the system should accept
- What *outputs* the system should produce
- What data the system should *store* that other systems might use
- What *computations* the system should perform
- The *timing and synchronization* of the above
Quality Requirements

- All must be verifiable
- Examples: Constraints on
  - Response time
  - Throughput
  - Resource usage
  - Reliability
  - Availability
  - Recovery from failure
  - Allowances for maintainability and enhancement
  - Allowances for reusability
Requirements Gathering Techniques

1. Use case analysis
2. Observation
3. Interviews
4. Brainstorming
5. Prototyping
Use case analysis

- Determine the classes of users that will use the facilities of this system (actors)
- Determine the tasks that each actor will need to do with the system
UML Use case diagrams

Registrar Actor

Add Course Offering

Add Course

Student

Enter Grade for Course

Professor Actor

Find information about course

Register in Course
Use-Cases: describing how the user will use the system

- A **use case** is a typical sequence of actions that a user performs in order to complete a given task
  - The objective of **use case analysis** is to model the system from the point of view of
    - how users interact with this system
    - when trying to achieve their objectives.
    - It is one of the key activities in requirements analysis
  - A **use case model** consists of
    - a set of use cases
    - an optional description or diagram indicating how they are related
Demonstration

- A use case describes what the customer wants the system to do

- Q: How do we or they know whether the system we have built can perform a given use case?

- A: We can **test** the use case

- That is, run the steps of the use case in a Java program, to demonstrate that the system performs as required.

- Such tests are called **acceptance tests for the system**
Interviewing and Client Goals

- Try to identify stakeholders’ **goals** as a first step:
  - What objectives are they trying to satisfy?

- Conduct a series of interviews
  - Ask about specific details
  - Ask about the stakeholder’s vision for the future
  - Ask if they have alternative ideas
  - Ask for other sources of information
  - Ask them to draw diagrams
Observation

- “Do what I do, don’t do what I say”
- Read documents and discuss requirements with users
- Shadowing important potential users as they do their work; ask the user to explain everything he or she is doing
- Session videotaping
Brainstorming

- Appoint an experienced moderator
- Arrange the attendees around a table
- Decide on a ‘trigger question’
- Ask each participant to write an answer and pass the paper to its neighbour
Prototyping

- Simplest kind: *paper prototype.*
  - a set of pictures of the system that are shown to users in sequence to explain what would happen

- Most common: a mock-up of the system’s UI
  - Written in a rapid prototyping language
  - Does *not* normally perform any computations, access any databases or interact with any other systems
  - May prototype a particular aspect of the system
Testable Requirements

- An acceptance test is a client’s test that the requirements have been met.
- Whenever you write a requirement, you should think about how you will check whether that requirement has been met.
- Requirements without tests may lead to serious disagreements with the client.
- Requirements without tests may not even be feasible to implement.
Acceptance test example

1. Alice, Bob and Charlie are 3 friends who have all registered with the MyFriends phone registry who send messages to one another on their mobile phones.
2. Alice successfully sends an SMS message “see you at 5pm”.
3. Bob checks his message count which is now 1.
4. When Bob reads message 1 he sees Alice’s “see you at 5pm” message.
5. Bob requests message id 5, but is alerted that there is no message with that id in his phone.
public static void main(String args[]) {

//1.Alice, Bob and Charlie are 3 friends who have all registered with the
//MyFriends phone registry who send messages to one another on their
//mobile phones.
    myfriends = new PhoneRegistry();
    alice = new MobilePhone("0111111111", 1005, myfriends);
    bob = new MobilePhone("0222222222", 400, myfriends);
    charlie = new MobilePhone("0333333333", 0, myfriends);
    if (myfriends.getRegistrySize() == 3) {
        System.out.println("1. Three phones successfully created");
    } else {
        failuremsg(1, "Failed to register 3 phones");
    }
}
//2. Alice successfully sends an SMS message “see you at 5pm”.
PhoneMessage msg1 = new PhoneMessage(
    alice.getNumber(),
    bob.getNumber(),
    "see you at 5pm"
);

boolean ok =
alice.sendSMS(msg1.getReceiver(),msg1.getMessage());
if (ok) {
    System.out.println("2. Alice to Bob SMS sent OK");
} else {
    failuremsg(2,"Alice sendSMS fails for: "+msg1.toString());
}
Heads Up: Requirements Difficulties
Requirements Difficulties

- Requirements change because
  - Business process changes
  - Technology changes
  - The problem becomes better understood

- Requirements analysis never stops
  - Continue to interact with the clients and users
  - The benefits of changes must outweigh the costs.
    - Certain small changes (e.g. look and feel of the UI) are usually quick and easy to make at relatively little cost.
    - Larger-scale changes have to be carefully assessed since forcing unexpected changes into a partially built system will probably result in a poor design and late delivery
  - Some changes are enhancements in disguise
    - Avoid making the system *bigger*, only make it *better*
Requirements Difficulties

- Lack of understanding of the domain or the real problem
  - Do domain analysis and prototyping

- Requirements change rapidly
  - Perform incremental development, build flexibility into the design, do regular reviews

- Attempting to do too much
  - Document the problem boundaries at an early stage, carefully estimate the time

- It may be hard to reconcile conflicting sets of requirements
  - Brainstorming, JAD sessions, competing prototypes

- It is hard to state requirements precisely
  - Break requirements down into simple sentences, review them carefully, look for potential ambiguity, make early prototypes