"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 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

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

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

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*

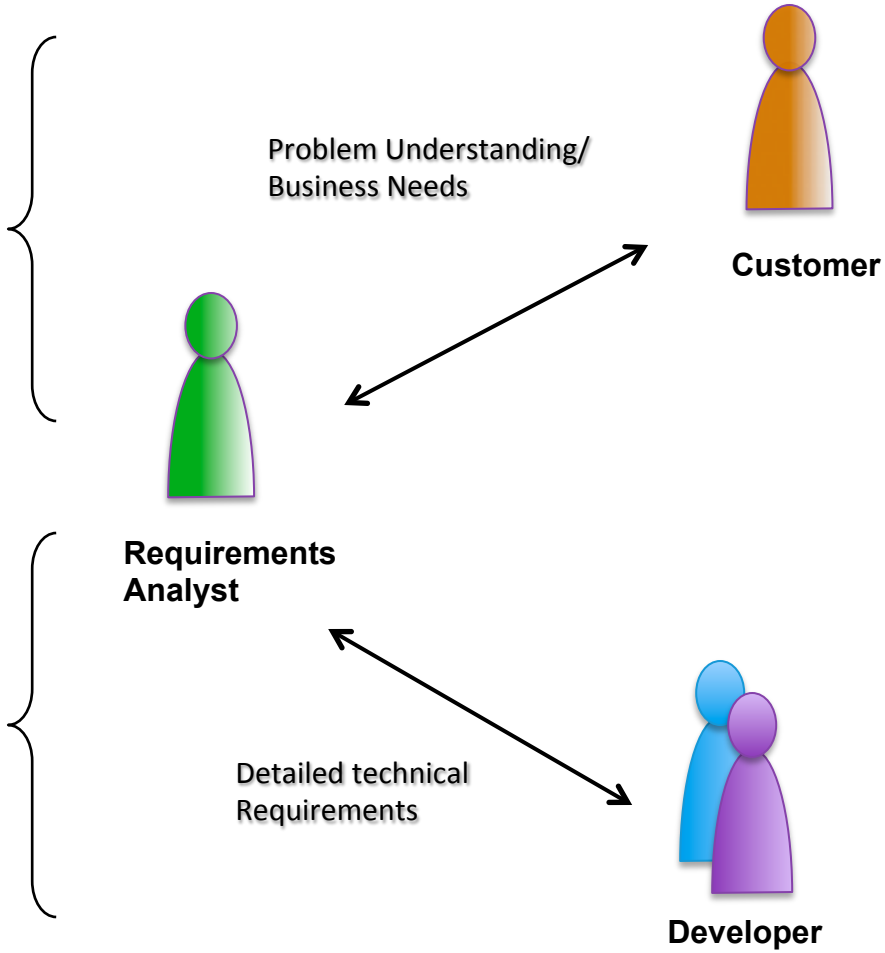
Needs of Different Audiences

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.



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

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.

Use Cases

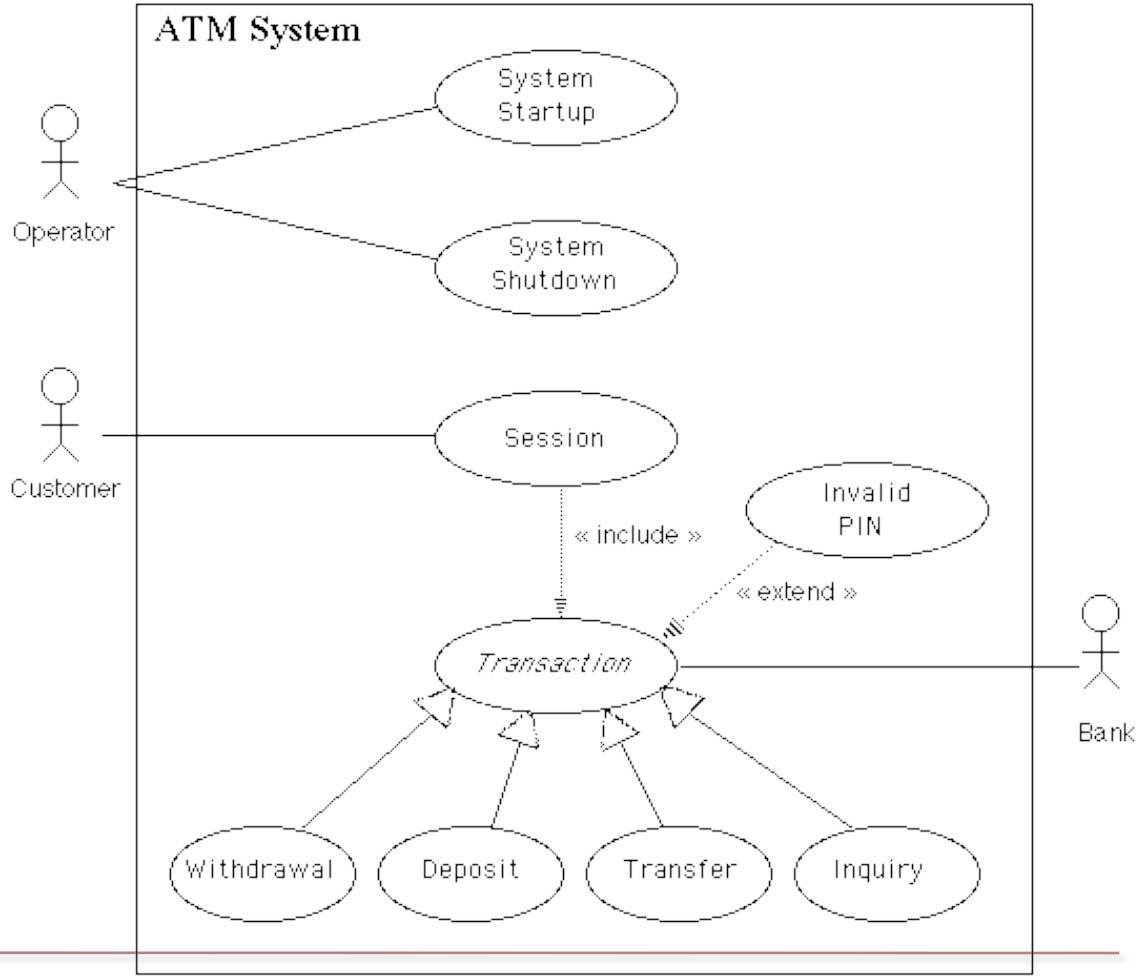
- Use Case: a story describing how the system and a user interact to accomplish a user task
- A form of User Centered Analysis – capturing requirements from the user's point of view
  - Goal of helping identify user needs
  - Solve the right problem
  - Describe the "business logic" of the system
- Use cases specify a *subset of functional requirements*
  - Only system behavior observable to the user
  - Does not address non-functional constraints, qualities
- Use cases should not specify design or implementation (including UI design)

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/



Creating Use Cases

- 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

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)

For Wednesday:

Prepare questions that help address your *biggest risks and uncertainties*

...but be prepared to improvise if some assumptions are not met

Consider multiple stakeholders/users:

- SafeRide user
- SafeRide driver
- SafeRide dispatcher
- SafeRide management (?)

Sketch use cases from each perspective. They will be wrong, but that's ok ... they will help you ask the right questions to discover what's wrong and how to make it right.