Starting Point for Software Development

- New development
- Evolution of existing system

<table>
<thead>
<tr>
<th>Requirements must be determined</th>
<th>Clients have produced requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>
Problem vs. Solution Views

- A problem can be expressed as:
  - A difficulty the users or customers are facing,
  - Or as an opportunity that will result in some benefit such as improved productivity or sales.
- Requirements documents describe problems

- The solution to the problem entails developing software
  - Software designs and their implementations in source code describe solutions

- Sometimes enforcing a sharp distinction between requirements and design can be counterproductive!

Overview

- Domain Modeling/System Engineering
- Requirements Analysis and Specification
- Describing Requirements using UML
- Pulling it all together: Using UML throughout the development process
System Engineering/Domain Analysis

- Software exists within some larger system
  - Encompassing system must be understood if software is to work properly within system
- The process by which a software engineer learns about the domain to better understand the problem is called domain analysis:
  - The domain is the general field of business or technology in which the clients will use the software
  - A domain expert is a person who has a deep knowledge of the domain
- System engineering is concerned with modeling the system encompassing software.
  - If the system exists within a business organization system engineering is referred to as business process engineering

Understanding the Context of a Software Application

- Who is the application for?
  - Identify stakeholders (customers, end-users)
- What problems will it solve?
  - Establish scope: determine features that will be in application and which will not
- Where will it be used?
  - Determine whether application is mission-critical, experimental, or a non-disruptive new capability, etc.
  - Understand how the application will fit in with other systems
- When is it needed?
  - Determine feasible time application can be developed and required time application is needed to meet business goals
- Why is it needed?
  - Build a business case for the application
- How will it work?
  - Brainstorm feasibility of solving the problem
Domain Analysis Document

A. Introduction
B. Glossary
C. General knowledge about the domain
D. Customers and users
E. The environment
F. Tasks and procedures currently performed
G. Competing software
H. Similarities to other domains

What is a Domain Model?

• Structuring of domain concepts
  – Identifies problem concepts and their relationships
• UML structural model used to depict structure
• Key Question: What are the objects of interest in this domain?
  – their attributes?
  – their relationships?
• IMPORTANT: This is a model of problem concepts; these concepts are NOT software objects, but a “visual dictionary” of domain concepts.
A Domain Model Does Not Represent Software Objects

An Example
System Engineering: Elements of a System

- **Software**: programs, models, and related documentation that model and implement desired services within system
- **Hardware**: Electronic devices that providing computing capability
- **People**: Users and operators of the systems
- **Database**: Organized collection of data used by system
- **Procedures**: Processes (sequences of steps) that define specific uses of the system within operating environment.

Business Process Engineering

- **Goal**: To describe processes that enable an organization to meet its goals.
- **Business process views**:
  - *Data view*: definition of data elements and their relationships
  - *Application portfolio*: identifies existing and planned programs/procedures and their relationships with each other
  - *Technology infrastructure*: hardware and software elements that are used to support the data and applications infrastructure
Software Requirements Engineering

Focus on eliciting, analyzing, documenting and managing requirements for the software component of a system.

• **Elicitation**: extracting requirements from customers and users
• **Analysis**: analyzing and organizing requirements to gain deeper understanding
  – Results of analysis reflected in models
• **Specification**: detailed documentation of required behavior
• **Validation**: requirements are validated against customer/user needs
• **Requirements management**: activities related to documenting, controlling and tracking requirements and changes to requirements

---

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 too broad
    • Determine high-level goals if too narrow

• Example: A university registration system

![Diagram](image-url)
What is a software requirement?

• A requirement is a statement about the proposed software that all stakeholders agree must be made true in order for the customer’s problem to be adequately solved.
  – Says something about what the software does and what data it maintains
  – All the stakeholders have agreed that it is valid
  – It helps solve the customer’s problem
• A collection of requirements is a requirements document.

Requirements types

• Functional requirements
  – Describe what the system should do
• Non-functional requirements
  – Constraints that must be adhered to during development
  – Examples: the software must restrict access to sensitive information; the software must deliver a response within a time interval after a particular event has occurred; the software must be usable by persons who are not computer literate.
Describing Functional Requirements

- What *inputs* the software should accept
- What *outputs* the software should produce
- What data the software should *store* that other systems might use
- What *computations* the software should perform
- What are the *timing and synchronization* of the above

Non-Functional Requirements

Three main types
- Categories reflecting: usability, efficiency, reliability, maintainability and reusability:
  - Response time, Throughput, Resource usage, Reliability, Availability, Recovery from failure, Allowances for maintainability and enhancement, Allowances for reusability
- Categories constraining the *environment and technology* of the software.
  - Platform
  - Technology to be used
- Categories constraining the *project plan and development methods*
  - Development process (methodology) to be used
  - Cost and delivery date
    - Often put in contract or project plan instead
Eliciting Requirements

- **Observation**
  - 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

- **Interviewing**
  - 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

Analysing Requirements

- **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

- *Joint Application Development (JAD) is a technique based on intensive brainstorming sessions*
Analysing Requirements (cont’d)

• Prototyping
  – The simplest kind: paper prototype.
    • a set of pictures of the system that are shown to users in sequence to explain what would happen
  – The 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

Requirements Specification & Analysis Tasks

• Brainstorm initial requirements statements
  – Develop use cases
  – Develop technical dictionary
• Document functional requirements and system attributes
• Develop exploratory prototypes
What is a use case model?

• use case model: a view of a system that emphasizes the behavior as it appears to outside users. A use case model partitions system functionality into transactions (‘use cases’) that are meaningful to users (‘actors’).

Use Case Analysis

• Identify and examine sources of information
  – talk to future users, application domain experts
  – collect and examine forms, process manuals, documentation of protocols etc.
• Information used to identify actors and use cases.
Simple Use Case Description Example

Use Case 1: Withdraw Money

Actor: Customer

Precondition: Customer has selected withdraw option.

Main flow:
1. System requests customer to enter identification.
2. Customer enters identification information.
3. If identification information is valid then the system acknowledges valid entry and asks for amount to be withdrawn.
4. Customer enters amount to be withdrawn.
5. If amount entered in less than amount in Customer’s account, system dispenses the cash, else system informs customer that amount cannot be withdrawn.
6. Use case ends.

Alternative flow: 3. If the identification is invalid then the use case is restarted. If this occurs 3 consecutive times then the system cancels the transaction and prevents the customer from interacting for 60 seconds.

Alternative flow: 2. The Customer can cancel the identification validation at anytime. Cancellation causes the use case to restart.
Identifying Use Cases and Actors

• Approaches to identifying use cases
  – *Actor-first*: Identify actors first and consider the ways they interact with the system
  – *Operation-first*: Identify system-level operations and then identify actors that interact with operations
  – *Event-first*: Identify external events and develop use cases that handle the events

Guidelines

• Some use cases do not have clear links to actors.
  – Focus on the event that causes the activity rather
• Focus on use cases in which actors get value from the interactions.
• Do not organize use cases along functional lines!
Describing Use Cases

- UML does not prescribe a format for use cases
- Use cases should describe the “should-be” state
  - Describing the “is” state can help in developing the “should-be” descriptions.
- Sometimes useful to associate user-interface prototypes with use cases.

Requirements Use Case Template - 1

<table>
<thead>
<tr>
<th>Use Case Number:</th>
<th>EU-xxxx: Indicates an essential use case, i.e., a use case that describes activity in system independent terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name:</td>
<td>Enter name of Use Case.</td>
</tr>
<tr>
<td>Overview:</td>
<td>Describe the purpose of the Use Case and give a brief description.</td>
</tr>
<tr>
<td>Type:</td>
<td>Enter Use Case priority (primary, secondary, optional)</td>
</tr>
<tr>
<td>Actors:</td>
<td>List all actors that participate in this Use Case. Indicate the actor that initiates the use case by placing “initiator” in brackets after the actor name. Also, indicate primary actors by placing “primary” in brackets after actor name.</td>
</tr>
<tr>
<td>Properties:</td>
<td>Performance:</td>
</tr>
<tr>
<td></td>
<td>Security:</td>
</tr>
<tr>
<td></td>
<td>Other:</td>
</tr>
</tbody>
</table>
Analysis Use Case Template – cont’d

| Pre condition: | Enter the condition that must be true when the main flow is initiated. This should reference the conceptual model. |
| Flow: | **Main Flow**: Steps should be numbered. |
| | **Subflows**: Break down of main flow steps |
| | **Alternate Flows**: Include the post condition for each alternate flow if different from the main flow. |
| Post Condition: | Enter the condition that must be true when the main flow is completed. This should reference the conceptual model. Include the following information in this section: |
| Cross References: | References to other Use Cases or textual requirements that relate to this Use Case. |

Technical Dictionary

- Terms used in the application domain are collected and defined in a *Technical Dictionary*.
- Minimizes *communication problems* between developers and clients. Problem examples:
  - Different terms used for the same concept
  - Same term used for different (sometime conflicting) concepts

© Robert B. France 3-31

© Robert B. France 3-32
Documenting Functional Requirements

- System functions: Services provided by software system
- Function categories
  - Evident: user-observable services that must be provided
  - Hidden: services that must be provided that are not visible to users (e.g., logging transactions in a banking system)
  - Frill: Optional services that enhance use of system

Function Specification Example

<table>
<thead>
<tr>
<th>Ref #</th>
<th>Function</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.1</td>
<td>Handle cash payments, capturing amount tendered and calculating balance due</td>
<td>evident</td>
</tr>
<tr>
<td>R1.2</td>
<td>Log credit payments</td>
<td>hidden</td>
</tr>
</tbody>
</table>
System Attributes/Non-Functional Requirements

- System attributes: characteristics or desired qualities of system
  - e.g., ease of use, fault tolerance, response time, ease of deployment
- May be applicable to all system functions or may apply only to a subset of system functions

Documenting Attributes of System Functions

<table>
<thead>
<tr>
<th>Ref #</th>
<th>Function</th>
<th>Cat.</th>
<th>Attribute</th>
<th>Details</th>
<th>Cat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0.9</td>
<td>Display desc. and item price</td>
<td>evident</td>
<td>Response time</td>
<td>5 secs. max</td>
<td>must</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interface</td>
<td>Forms based graphical</td>
<td></td>
</tr>
<tr>
<td>R1.2</td>
<td>Log credit payments</td>
<td>hidden</td>
<td>Fault tolerance</td>
<td>Must log within 24 hrs even in power failure</td>
<td>must</td>
</tr>
</tbody>
</table>
Exploratory Prototypes

- A prototype is a sequence of dialog designs.
- A dialog design is user-interface mock-up that allows user to enter data.
- Prototypes are used to validate use cases with clients.
- Dialog designs can be diagrammatically associated with use cases.

Pulling it all together

Using UML models throughout the development process
An “UltraLite” UML Process

A “Full” UML Process
Another Approach to Requirements Analysis: Quality Function Deployment

• Quality management technique that transforms needs into technical requirements

• Types of requirements
  – Normal: customer is satisfied if these requirements are met
  – Expected: implicit requirements that are fundamental needs; absence will result in dissatisfaction
  – Exciting: go beyond customer’s expectations

QFD (cont’d)

• Function deployment: determine value of required functions (services)
• Information deployment: identify data elements and events that are consumed and produced by system
• Task deployment: analysis of product behavior within operating environment
• Value analysis: determine relative priority of requirements; determined in above deployments