Lecture # 41-42

Review of Lectures 23-40

Writing Requirements

Writing Requirements

- Requirements specification should establish an understanding between customers and suppliers about what a system is supposed to do, and provide a basis for validation and verification

Writing Requirements

- Typically, requirements documents are written in natural languages (like, English, Japanese, French, etc.)
- Natural languages, by their nature, are ambiguous
- Structured languages can be used with the natural languages to specify requirements

Guidelines for Writing Requirements

- Define standard templates for describing requirements
- Use language simply, consistently, and concisely
- Use diagrams appropriately

Use of Standard Templates

- Define a set of standard format for different types of requirements and ensure that all requirement definitions adhere to that format
- Standardization means that omissions are less likely and makes requirements easier to read and check

Requirements Document

Requirements Document

- The requirements document is a formal document used to communicate the requirements to customers, engineers and managers
- It is also known as software requirements specifications or SRS

Requirements Document

- The services and functions which the system should provide
- The constraints under which the system must operate
- Overall properties of the system i.e., constraints on the system's emergent properties

Requirements Document

- Definitions of other systems which the system must integrate with
- Information about the application domain of the system, e.g., how to carry out particular types of computation
- Constraints on the process used to develop the system

Requirements Document

- It should include both the user requirements for a system and a detailed specification of the system requirements
- In some cases, the user and system requirements may be integrated into one description, while in other cases user requirements are described before (as introduction to) system requirements

Requirements Document

- Typically, requirements documents are written in natural languages (like, English, Japanese, French, etc.)
- Natural languages, by their nature, are ambiguous
- Structured languages can be used with the natural languages to specify requirements

Requirements Document

- For software systems, the requirements document may include a description of the hardware on which the system is to run
- The document should always include an introductory chapter which provides an overview of the system and the business needs

Requirements Document

- A glossary should also be included to document technical terms
- And because multiple stakeholders will be reading documents and they need to understand meanings of different terms
- Also because stakeholders have different educational backgrounds

Requirements Document

- Structure of requirements document is also very important and is developed on the basis of following information
  - Type of the system
  - Level of detail included in requirements
  - Organizational practice
  - Budget and schedule for RE process

Users of Requirements Documents

- System customers
- Managers
- System engineers
- System test engineers
- System maintenance engineers

Quality Attributes of Requirements Documents

SRS Quality Attributes

- Correct
- Unambiguous
- Complete
- Verifiable
- Consistent

SRS Quality Attributes

- Understandable by customer
- Modifiable
  - Traced
  - Traceable
  - Design independent

SRS Quality Attributes

- Annotated
- Concise
- Organized

Use Case Modeling

Introduction

- No system exists in isolation. Every interesting system interacts with human or automated actors that use that system for some purpose, and those actors expect that system to behave in predictable ways

Introduction

- A use case specifies the behavior of a system or part of a system
- It is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor

Introduction

- They are used in requirements elicitation process
- Use cases are applied to capture the intended behavior of the system under development, without having to specify how the behavior is implemented
- They provide a way for developers, end users, and domain experts to come to a common understanding

Introduction

- Use cases serve to help validate the architecture and to verify the system as it evolves during development
- Use cases are realized by collaborations whose elements work together to carry out each use case

Introduction

- Well-structured use cases denote essential system or subsystem behaviors only
- They are neither overly general nor too specific

Use Case Modeling

Documenting Use Cases

- Name
- Summary
  - Short description of use case
- Dependency (on other use cases)
- Actors
- Preconditions
  - Conditions that are true at start of use case

Documenting Use Cases

- Flow of Events
  - Narrative description of basic path
- Alternatives
  - Narrative description of alternative paths
- Post-condition
  - Condition that is true at end of use case

Case Study and Use Cases for Case Study

A Well-Structured Use Case

- Names a single, identifiable, and reasonably atomic behavior of the system or part of the system
- Factors common behavior by pulling such behavior from other use cases
- Factors variants by pushing such behavior into other use cases that extend it

Banking System Case Study

Problem Description - 1

- A bank has several automated teller machines (ATMs), which are geographically distributed and connected via a wide area network to a central server. Each ATM machine has a card reader, a cash dispenser, a keyboard/display, and a receipt printer. By using the ATM machine, a customer can withdraw cash from either checking or savings account, query the balance of an account, or transfer funds from one account to another. A transaction is initiated when a customer inserts an ATM card into the card reader. Encoded on the magnetic strip on the back of the ATM card are the card number, the start date, and the expiration date. Assuming the card is recognized, the system validates the ATM card to determine that the expiration date has not passed, that the user-entered PIN (personal identification number) matches the PIN maintained by the system, and that the card is not lost or stolen. The customer is allowed three attempts to enter the correct PIN; the card is confiscated if the third attempt fails. Cards that have been reported lost or stolen are also confiscated.

Problem Description - 2

- If the PIN is validated satisfactorily, the customer is prompted for a withdrawal, query, or transfer transaction. Before withdrawal transaction can be approved, the system determines that sufficient funds exist in the requested account, that the maximum daily limit will not be exceeded, and that there are sufficient funds available at the local cash dispenser. If the transaction is approved, the requested amount of cash is dispensed, a receipt is printed containing information about the transaction, and the card is ejected. Before a transfer transaction can be approved, the system determines that the customer has at least two accounts and that there are sufficient funds in the account to be debited. For approved query and transfer requests, a receipt is printed and card ejected. A customer may cancel a transaction at any time; the transaction is terminated and the card is ejected. Customer records, account records, and debit card records are all maintained at the server.

Problem Description - 3

- An ATM operator may start up and close down the ATM to replenish the ATM cash dispenser and for routine maintenance. It is assumed that functionality to open and close accounts and to create, update, and delete customer and debit card records is provided by an existing system and is not part of this problem.

- 'Designing Concurrent, Distributed, and Real-Time Applications with UML' by H. Gomaa, Addison-Wesley, 2000

Use Case Model

- The use cases are described in the use case model
- There are two actors of this system
  - ATM Customer
  - Operator

Need for Modeling

- Modeling is a central part of all activities that lead up to the development good software
- We build models to communicate the desired structure and behavior of our system

Need for Modeling

- We build models to visualize and control the system's architecture
- We build models to better understand the system we are building, often exposing opportunities for simplification and reuse
- We build models to minimize risk

Need for Modeling

- We build models so that we can better understand the system we are developing
- We build models of complex systems because we cannot comprehend such a system in its entirety

A model is a simplification of reality

Four Aims of Modeling

- Models help us to visualize a system as it is or as we want it to be
- Models permit us to specify the structure or behavior of a system
- Models give us a template that guides us in constructing a system
- Models document the decisions we have made

Principles of Modeling

- The choice of what models to create has profound influence on how a problem is attacked and how a solution is shaped
- Every model may be expressed at different levels of precision

Principles of Modeling

- The best models are connected to reality
- No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models

Principles of Modeling

- Partitioning
- Abstraction
- Projection

Modeling Techniques

- Object-oriented modeling
  - Static and dynamic modeling
- Functional modeling
- Dynamic modeling

Object-Oriented Modeling with UML

Static Structure

- Any precise model must first define the universe of discourse, that is, the key concepts from the application, their internal properties, and their relationships to each other
- This set of constructs is the static view

Static Structure

- The application concepts are modeled as classes, each of which describes a set of discrete objects that hold information and communicate to implement behavior
- The information they hold is modeled as attributes; the behavior they perform is modeled as operations

Object-Oriented Static Modeling of the Banking System

Steps in Object-Oriented Analysis

- Identify classes within the problem domain
- Define the attributes and methods of these classes
- Define the behavior of those classes
- Model the relationship between those classes

Object-Oriented Static Modeling of the Banking System

Object-Oriented Static Modeling of the Banking System

Functional Modeling

Function-oriented Modeling

- In function-oriented modeling, a hierarchy of functions (also known as processes, transforms, transactions, bubbles, and activities) is created
- At the root of the hierarchy is the most abstract function, while the leaf nodes of the hierarchy are least abstract

Function-oriented Modeling

- Function-oriented modeling is based on the concept of functions or processes, so they become the most important element in this approach
- The functional model describes computations within a system, i.e., what happens

Function-oriented Modeling

- What is a function or a transform or process?
- Each function is a sequential activity that potentially may execute concurrently with other functions
- The functional model specifies the result of a computation without specifying how or when they are computed

Types of Functions

- Asynchronous Function
- Asynchronous State Dependent Function
- Periodic Function
- Periodic State-Dependent Function

Real-Time Structured Analysis and Design Technique (RSTAD) and Application of Banking Case Study

Function-oriented Modeling Techniques

- Structured requirements definition
- Structured analysis and system specification
- Modern structured analysis
- **Real-time structured analysis and structured design**
- **Structured analysis and design technique**
- PSL/PSA

Real-Time Structured Analysis and Structured Design (RSTAD)

- Develop the system context diagram
- Perform data flow/control flow decomposition
- Develop control transformations or control specifications
- Define mini-specifications (process specifications)
- Develop data dictionary

Structured Analysis and Design Technique

Structured Analysis and Design Technique (SADT)

- A model of the problem is constructed, which is composed of hierarchy of diagrams
- Each diagram is composed of boxes and arrows
- The topmost diagram, called the context diagram, defines the problem most abstractly

Structured Analysis and Design Technique (SADT)

- As the problem is refined into sub-problems, this refinement is documented into other diagrams
- Boxes should be given unique names that should always be verb phrases, because they represent functions

Structured Analysis and Design Technique (SADT)

- All boxes should be numbered in the lower right corner, to reflect their relative dominance
- Arrows may enter top, left, or bottom sides of the box, and can exit only from the right side of the box

Structured Analysis and Design Technique (SADT)

- An arrow pointing into the left side of a box represents things that will be *transformed* by the box. These are inputs
- An arrow pointing down into the top of the box represents *control* that affects how the box transforms the things entering from left side

Structured Analysis and Design Technique (SADT)

- Arrows entering the bottom of a box represent *mechanism* and provide the analyst with the ability to document how the function will operate, who will perform it, or what physical resources are needed to perform the function

Dynamic Modeling

Dynamic Modeling

- Temporal relationships are difficult to understand
- A system can first be understood by examining its static structure and the relationships that exist among different elements at single moment in time

Dynamic Modeling

- Then we examine changes in these elements and their relationships over time
- Those aspects of a system that are concerned with time and changes are the dynamic model

Dynamic Modeling

- Control is an aspect of a system that describes the sequences of operations that occur in response to external stimuli, without consideration of what the operations do, what they operate on, or how they are implemented

Dynamic Modeling

- Dynamic modeling deals with flow of control, interactions, and sequencing of operations in a system of concurrently-active objects
- Major dynamic modeling concepts are *events*, which represent external stimuli, and *states*, which represent values of objects

Dynamic Modeling

- There are two ways to model dynamic behavior
- One is the life history of one object as it interacts with the rest of the world; the other is the communication patterns of a set of connected objects as they interact to implement behavior

Dynamic Modeling

- The view of an object in isolation is a state machine – a view of an object as it responds to events based on its current state, performs actions as part of its response, and transitions to a new state
- This is displayed in state chart diagrams in UML

Dynamic Modeling

- The view of a system of interacting objects is a collaboration, a context-dependent view of objects and their links to each other, together with the flow of messages between objects across data links
- Collaboration and sequence diagrams are used for this view in UML

Techniques for Dynamic Modeling

- Finite state machines (FSM)
- Statecharts
- Petri nets
- Decision tables and decision trees
- Collaboration diagrams
- Sequence diagams

Dynamic Modeling of Banking System Case Study

Dynamic Modeling

- There are two ways to model dynamic behavior
- One is the life history of one object as it interacts with the rest of the world; the other is the communication patterns of a set of connected objects as they interact to implement behavior

Dynamic Modeling

- The view of an object in isolation is a state machine – a view of an object as it responds to events based on its current state, performs actions as part of its response, and transitions to a new state
- This is displayed in state chart diagrams in UML

Dynamic Modeling

- The view of a system of interacting objects is a collaboration, a context-dependent view of objects and their links to each other, together with the flow of messages between objects across data links
- Collaboration and sequence diagrams are used for this view in UML

Dynamic Modeling

- The dynamic model depicts the interaction among the objects that participate in each use case
- The starting point for developing the dynamic model is the use case and the objects determined during object structuring

Dynamic Modeling of Banking System Case Study

Dynamic Modeling

- There are two ways to model dynamic behavior
- One is the life history of one object as it interacts with the rest of the world; the other is the communication patterns of a set of connected objects as they interact to implement behavior

Dynamic Modeling

- The view of an object in isolation is a state machine – a view of an object as it responds to events based on its current state, performs actions as part of its response, and transitions to a new state
- This is displayed in state chart diagrams in UML

Dynamic Modeling

- The view of a system of interacting objects is a collaboration, a context-dependent view of objects and their links to each other, together with the flow of messages between objects across data links
- Collaboration and sequence diagrams are used for this view in UML. Both of these combined are called interactive diagrams

Dynamic Modeling

- The dynamic model depicts the interaction among the objects that participate in each use case
- The starting point for developing the dynamic model is the use case and the objects determined during object structuring

Interaction Diagrams

- Interaction diagrams are used to model the dynamic aspects a system. For the most part, this involves modeling concrete or prototypical instances of classes, interfaces, components, and nodes, along with messages that are dispatched among them, all in the context of a scenario that illustrates a behavior

Interaction Diagrams

- Interaction diagrams may stand alone to visualize, specify, construct, and document the dynamics of a particular society of objects, or they may be used to model one particular flow of control of a use case

Types of Interaction Diagrams

- Sequence diagrams
- Collaboration diagrams

Sequence Diagrams

- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages
- Graphically, a sequence diagram is a table that shows objects arranged along the X axis and messages, ordered in increasing time, along the Y axis

Collaboration Diagrams

- A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages
- Graphically, a collaboration diagram is a collection of vertices and arcs

Hints and Tips on Interaction Diagrams

- Give it a name that communicates its purpose
- Use a sequence diagram if you want to emphasize the time ordering of messages
- Use a collaboration diagram if you want to emphasize the organization of the objects involved in the interaction

Hints and Tips on Interaction Diagrams

- Lay out its elements to minimize lines that cross
- Use notes and color as visual cues to draw attention to important features of your diagram

Requirements Document for the Banking System

Requirements Document

- The requirements document is a formal document used to communicate the requirements to customers, engineers and managers
- It is also known as software requirements specifications or SRS

Requirements Document

- The services and functions which the system should provide
- The constraints under which the system must operate
- Overall properties of the system i.e., constraints on the system's emergent properties

SRS for the Banking System

- Preface
- Introduction
- Glossary
- Specific requirements
- Appendices
  - Use-case model
  - Object model
  - Data-flow model

Requirements Prioritization

Introduction

- In everyday life, we make decisions, e.g., when buying a DVD-player, food, a telephone, etc
- Usually, we do not have more than a couple of choices to consider. Even with just a couple of choices, decisions can be difficult to make

Introduction

- When having tens, hundreds or even thousands of alternatives, decision-making becomes much more difficult
- One of the keys to making the right decision is to prioritize between different alternatives. It is often not obvious which choice is better, because several aspects must be taken into consideration

Introduction

- When developing software systems, similar trade-offs must be made
- The functionality that is most important for the customers might not be as important when other aspects (e.g. price) are factored in
- We need to develop the functionality that is most desired by the customers, as well as least risky, least costly, and so forth

Introduction

- The quality of software products is often determined by the ability to satisfy the needs of the customers and users
- So, it is very important to include those requirements in the product, which are really needed by the customers

Introduction

- Most software projects have more candidate requirements than can be realized within the time and cost constraints
- Prioritization helps to identify the most valuable requirements from this set by distinguishing the critical few from the trivial many

Requirements Prioritization

- Act of giving precedence or priority to one item over another item
- Requirements prioritization means giving precedence to some requirements over other requirements based on feedback from system stakeholders

Benefits of Requirements Prioritization

- Stakeholders can decide on the core requirements for the system
- Planning and selection of ordered, optimal set of software requirements for implementation in successive releases
- Helps in trade-offs of conflicting constraints such as schedule, budget, resources, time to market, and quality

Benefits of Requirements Prioritization

- Balances the business benefit of each requirement against its cost
- Balances the implications of requirements on the software architecture and future evolution of the product and its associated cost
- Selects only a subset of the requirements and still produce a system that will satisfy the customers

Benefits of Requirements Prioritization

- Get technical advantage and optimize market opportunity
- Minimize rework and schedule slippage (plan stability)
- Handle contradictory requirements, focus the negotiation process, and resolve disagreements between stakeholders

Benefits of Requirements Prioritization

- Establish relative importance of each requirement to provide the greatest value at the lowest cost

Research Areas in Requirements Engineering

Topics in Requirement Engineering

- Requirements elicitation techniques
- Requirements validation techniques
- Requirements management and traceability
- Requirements evolution
- Requirements and software architectures

Topics in Requirement Engineering

- Requirements and business architectures
- Requirements prioritizing and negotiation
- Specification of quality attributes
- Requirements metrics
- Tool support for RE

Topics in Requirement Engineering

- Cognitive, social and cultural factors in RE

Thank you