Prof. Dr. Nizamettin AYDIN

naydin@yildiz.edu.tr
http://www.yildiz.edu.tr/~naydin

1

Object-Oriented Systems Analysis and Design with the UML

Objectives:

- Understand the basic characteristics of object-oriented systems.
- Be familiar with the Unified Modeling Language (UML), Version 2.0.
- Be familiar with the Unified Process.
- Understand a minimalist approach to object-oriented systems analysis and design.

- Until recent years, analysts focused on either data or business processes when developing systems.
- As they moved through the SDLC, they emphasized either
  - the data for the system (data-centric approaches)
  - the processes that it would support (process-centric approaches).
- In the mid-1980s, developers were capable of building systems that could be more efficient
  - if the analyst worked with a system's data and processes simultaneously and focused on integrating the two.
- As such, project teams began using an object-oriented approach,
  - whereby self-contained modules called objects (containing both data and processes) were used as the building blocks for systems.

3

History of object-oriented approaches

- Simula, an OOP language, was created in the 1960s
- Smalltalk was created in the early 1970s.
- Until the mid-1980s, developers had to keep the data and processes separate
  - to be capable of building systems that could run on the mainframe computers of that era.
- Today, due to the increase in processor power and the decrease in processor cost, OO approaches are feasible.

4

Basic Characteristics of Object Oriented Systems

- OO systems focus on capturing the structure and behavior of information systems in little modules that encompass both data and process.
- These little modules are known as objects.
- The basic characteristics of OO systems include
  - classes,
  - objects,
  - methods,
  - messages,
  - encapsulation,
  - information hiding,
  - inheritance,
  - polymorphism,
  - dynamic binding.

5

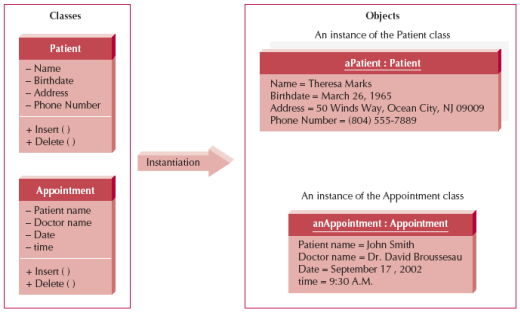
Classes and Objects

- Class
- the general template we use to define and create specific instances, or objects.
- Every object is associated with a class.
  - For example, all of the objects that capture information about patients could fall into a class called Patient, because there are
    - attributes (e.g., names, addresses, and birth dates)
    - methods (e.g., insert new instances, maintain information, and delete entries)

    that all patients share (see next Figure).

6

Classes and Objects



7

Classes and Objects

- Object
  - an instantiation of a class.
  - a person, place, event, or thing about which we want to capture information.
- If we were building an appointment system for a doctor's office, classes might include doctor, patient, and appointment.
- The specific patients like Jim Maloney, Mary Wilson, and Theresa Marks are considered instances, or objects, of the patient class.

8

Classes and Objects

- Each object has attributes that
  - describe information about the object, such as
    - a patient's name, birth date, address, and phone number.
  - The state of an object is defined by the value of its attributes and its relationships with other objects at a particular point in time.
    - For example, a patient might have a state of "new" or "current" or "former."
- Each object also has behaviors that
  - specify what the object can do.
    - For example, an appointment object likely can schedule a new appointment, delete an appointment, and locate the next available appointment.

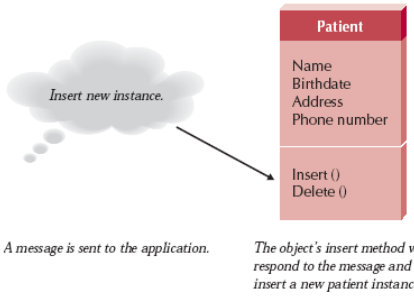
9

Methods and Messages

- Methods implement an object's behavior.
  - nothing more than an action that an object can perform.
  - analogous to a function or procedure in a traditional programming language such as C, Cobol, or Pascal.
- Messages are information sent to objects to trigger methods.
  - It is a function or procedure call from one object to another object.
    - For example, if a patient is new to the doctor's office, the system will send an insert message to the application. The patient object will receive a message (instruction) and do what it needs to do to go about inserting the new patient into the system (see next Figure).

10

Methods and Messages



A message is sent to the application.

The object's insert method will respond to the message and insert a new patient instance.

11

Encapsulation and Information Hiding

- Encapsulation
  - the combination of process and data into a single entity.
- Traditional approaches to information systems development tend to be either
  - process-centric
    - e.g., structured systems

  or
  - data-centric
    - e.g., information engineering.
- Object-oriented approaches combine process and data into holistic entities (objects).

12

Encapsulation and Information Hiding

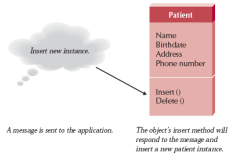
- Information hiding
  - first promoted in structured systems development.
  - only the information required to use a software module is published to the user of the module.
    - This implies the information required to be passed to the module, and the information returned from the module is published.
    - How the module implements the required functionality is not relevant.
- In OO systems, combining encapsulation with the information hiding principle suggests that the information hiding principle be applied to objects instead of merely applying it to functions or processes.
  - Objects are treated like black boxes.

13

Encapsulation and Information Hiding

- Reusability Key
  - Use an object by calling methods
    - because it shields the internal workings of the object from changes in the outside system, and it keeps the system from being affected when changes are made to an object.

- In the following figure, notice how a message (insert new patient) is sent to an object yet the internal algorithms needed to respond to the message are hidden from other parts of the system.
- The only information that an object needs to know is the set of operations, or methods, that other objects can perform and what messages need to be sent to trigger them.



14

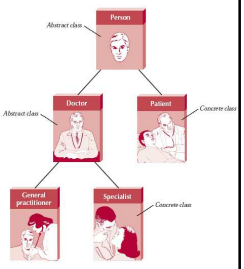
Inheritance

- Inheritance,
  - was proposed in data modeling in the late 1970s and the early 1980s.
- The data modeling literature suggests using inheritance to identify higher-level, or more general, classes of objects.
- Common sets of attributes and methods can be organized into superclasses.
- Typically, classes are arranged in a hierarchy whereby
  - the superclasses, or general classes, are at the top,
  - the subclasses, or specific classes, are at the bottom.

15

Inheritance

- In the following figure,
  - person is a superclass to the classes Doctor and Patient.
  - Doctor, in turn, is a superclass to general practitioner and specialist.
- Notice how a class (e.g., doctor) can serve as a superclass and subclass concurrently.
- The relationship between the class and its superclass is known as the A-Kind-Of (AKO) relationship.
  - For example, in the figure,
    - a general practitioner is A-Kind-Of doctor, which is A-Kind-Of person.



16

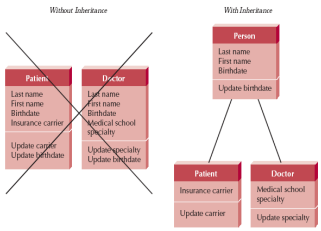
Inheritance

- Subclasses inherit the appropriate attributes and methods from the superclasses above them.
  - That is, each subclass contains attributes and methods from its parent superclass.
    - For example, previous figure shows that both doctor and patient are subclasses of person and therefore will inherit the attributes and methods of the person class.
  - Inheritance makes it simpler to define classes.
    - In previous figure, instead of repeating the attributes and methods in the doctor and patient classes separately, the attributes and methods that are common to both are placed in the person class and inherited by those classes below it.

17

Inheritance

- Notice how much more efficient hierarchies of object classes are than the same objects without a hierarchy in the following figure.



18

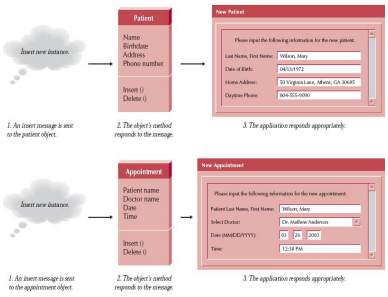
Polymorphism and Dynamic Binding

- Polymorphism means that
  - the same message can be interpreted differently by different classes of objects.
    - For example, inserting a patient means something different than inserting an appointment.
    - As such, different pieces of information need to be collected and stored.
- not have to be concerned with how something is done when using objects.
- simply send a message to an object, and that object will be responsible for interpreting the message appropriately.

19

Polymorphism and Dynamic Binding

- For example, if we sent the message "Draw yourself " to a square object, a circle object, and a triangle object, the results would be very different, even though the message is the same.
- Notice in Figure how each object responds appropriately (and differently) even though the messages are identical.



20

Polymorphism and Dynamic Binding

- Polymorphism is made possible through dynamic binding (late binding)
  - a technique that delays typing the object until run-time.
    - As such, the specific method that is actually called is not chosen by the object-oriented system until the system is running.
- This is in contrast to static binding, in which
  - the type of object would be determined at compile time.
  - The developer would have to choose which method should be called instead of allowing the system to do it.
    - This is why in most traditional programming languages you find complicated decision logic based on the different types of objects in a system.

21

Polymorphism and Dynamic Binding

- For example, in a traditional programming language, instead of sending the message "Draw yourself " to the different types of graphical objects in the previous figure, you would have to write decision logic using a case statement or a set of if statements to determine what kind of graphical object you wanted to draw, and you would have to name each draw function differently (e.g., drawsquare, draw-circle, or draw-triangle).
- This obviously would make the system much more complicated and more difficult to understand.

22

The Unified Modeling Language, Version 2.0

Until 1995, object concepts were popular

- but implemented in many different ways by different developers.
- Each developer had his or her own methodology and notation.

Rational Software brought three industry leaders together to create a single approach to object-oriented systems development.

- Grady Booch, Ivar Jacobson, and James Rumbaugh worked with others to create a standard set of diagramming techniques known as the Unified Modeling Language (UML).

The Unified Modeling Language

- The objective of UML
  - to provide a common vocabulary of OO terms and diagramming techniques that is
    - rich enough to model any systems development project from analysis through implementation.
- In November 1997, the Object Management Group (OMG) formally accepted UML as the standard for all object developers.
- Over the years since, the UML has gone through multiple minor revisions.

24

The Unified Modeling Language

- The Version 2.0 of the UML defines a set of fourteen diagramming techniques used to model a system.
- Version history:
  - UML 2.0 major revision was adopted by the OMG in 2005
  - UML 2.1.1 and UML 2.1.2 appeared in 2007
  - UML 2.2 was released in February 2009.
  - UML 2.3 was formally released in May 2010.
  - UML 2.4.1 was formally released in August 2011.

25

The Unified Modeling Language

- The diagrams are broken into two major groupings:
- one for modeling structure of a system
- The structure modeling diagrams include
  - class,
  - object,
  - package,
  - deployment,
  - component,
  - composite structure diagrams.

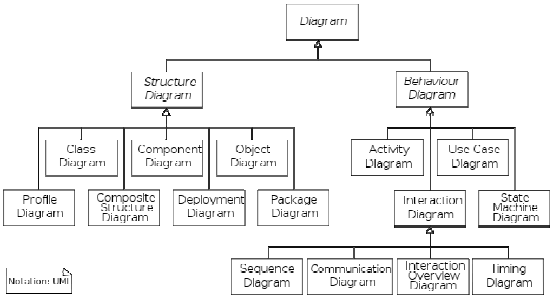
26

The Unified Modeling Language

- one for modeling behavior.
- The behavior modeling diagrams include
  - activity,
  - sequence,
  - communication,
  - interaction overview,
  - timing,
  - behavior state machine,
  - protocol state machine,
  - use case diagrams.

27

The Unified Modeling Language



28

The Unified Modeling Language

- Depending on where in the development process the system is, different diagrams play a more important role.
- In some cases, the same diagramming technique is used throughout the development process.
  - In that case, the diagrams start off very conceptual and abstract.
  - As the system is developed, the diagrams evolve to include details that ultimately lead to code generation and development.

29

Structure diagrams

- emphasize the things that must be present in the system being modeled.
- used extensively in documenting the software architecture of software systems.
- Class diagram (Analysis, Design)
  - describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.
- Component diagram (Physical Design, Implementation)
  - describes how a software system is split up into components and shows the dependencies among these components.

30

Structure diagrams

- Composite structure diagram (Analysis, Design)
  - describes the internal structure of a class and the collaborations that this structure makes possible.
- Deployment diagram (Physical Design, Implementation)
  - describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.
- Object diagram (Analysis, Design)
  - shows a complete or partial view of the structure of an example modeled system at a specific time.

31

Structure diagrams

- Package diagram (Analysis, Design, Implementation)
  - describes how a system is split up into logical groupings by showing the dependencies among these groupings.
- Profile diagram
  - operates at the metamodel level to show stereotypes as classes with the <<stereotype>> stereotype, and profiles as packages with the <<profile>> stereotype.
  - The extension relation (solid line with closed, filled arrowhead) indicates what metamodel element a given stereotype is extending.

32

Behavior Diagrams

- emphasize what must happen in the system being modeled.
- used extensively to describe the functionality of software systems.
- Activity diagram (Analysis, Design)
  - describes the business and operational step-by-step workflows of components in a system.
  - An activity diagram shows the overall flow of control.
- UML state machine diagram (Analysis, Design)
  - describes the states and state transitions of the system.

33

Behavior Diagrams

- Use Case Diagram (Analysis)
  - describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.
- a subset of behavior diagrams is Interaction diagrams
  - emphasize the flow of control and data among the things in the system being modeled
- Timing diagrams (Analysis, Design)
  - a specific type of interaction diagram where the focus is on timing constraints.

34

Interaction diagrams

- Communication diagram (Analysis, Design)
  - shows the interactions between objects or parts in terms of sequenced messages.
  - represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.
- Interaction overview diagram (Analysis, Design)
  - provides an overview in which the nodes represent communication diagrams.

35

Interaction diagrams

- Sequence diagram (Analysis, Design)
  - shows how objects communicate with each other in terms of a sequence of messages.
  - Also indicates the lifespans of objects relative to those messages.

36

UML 2.0 Diagram Summary

Diagram Name	Used to	Primary Phase
Structure Diagrams		
Class	Illustrate the relationships between classes modeled in the system.	Analysis, Design
Object	Illustrate the relationships between objects modeled in the system. Used when actual instances of the classes will better communicate the model.	Analysis, Design
Package	Group other UML elements together to form higher level constructs.	Analysis, Design, Implementation
Deployment	Show the physical architecture of the system. Can also be used to show software components being deployed onto the physical architecture.	Physical Design, Implementation
Component	Illustrate the physical relationships among the software components.	Physical Design, Implementation
Composite Structure	Illustrate the internal structure of a class, i.e., the relationships among the parts of a class.	Analysis, Design
Behavioral Diagrams		
Activity	Illustrate business workflows independent of classes, the flow of activities in a use case, or detailed design of a method.	Analysis, Design
Sequence	Model the behavior of objects within a use case. Focuses on the time-based ordering of an activity.	Analysis, Design
Communication	Model the behavior of objects within a use case. Focuses on the communication among a set of collaborating objects of an activity.	Analysis, Design
Interaction Overview	Illustrate an overview of the flow of control of a process.	Analysis, Design
Timing	Illustrate the interaction that takes place among a set of objects and the state changes in which they go through along a time axis.	Analysis, Design
Behavioral State Machine	Examine the behavior of one class.	Analysis, Design
Protocol State Machine	Illustrates the dependencies among the different interfaces of a class.	Analysis, Design
Use-Case	Capture business requirements for the system and to illustrate the interaction between the system and its environment.	Analysis

37

Extension Mechanisms

- As large and as complete as the UML is, it is impossible for the creators of the UML to anticipate all potential uses.
- Therefore UML also provides a set of extension mechanisms. These include
  - stereotypes,
  - tagged values,
  - constraints,
  - profiles.

38

Extension Mechanisms

- Stereotypes
  - provide the analyst with the ability to incrementally extend the UML using the model elements already in the UML.
- A stereotype is shown as a text item enclosed within guillemets (<< >>) or angle brackets (<< >>).
- Stereotypes can be associated with any model element (e.g., class, object, use case, relationships) within any UML diagram.

39

Extension Mechanisms

- Tagged Values
  - In the UML, all model elements have properties that describe them.
  - Tagged values are used to add new properties to a base element.
    - For example, if a project team was interested in tracing the authorship of each class in a class diagram, the project team could extend the class element to include an author property.
  - It is also possible to associate tagged values with specific stereotypes.
    - In this manner, when the analyst applies a stereotype to a model element, all of the additional tagged values associated with the stereotype also are applied.

40

Extension Mechanisms

- Constraints
  - allow the analyst to model problem domain specific semantics by placing additional restrictions on the use of model elements.
  - Constraints are typically modeled using the Object Constraint Language (OCL).
- Profiles
  - allow the developer to group a set of model elements that have been extended using stereotypes, tagged values, and/or constraints into a package.
  - have been used to create modeling extensions that can address specific types of implementation platforms, such as .NET, or specific modeling domains, such as embedded systems.

41

Object-Oriented Systems Analysis and Design

OO approaches to developing IS can use any of the traditional methodologies (waterfall development, parallel development, phased development, prototyping, and throwaway prototyping).

However, the OO approaches are most associated with a phased development RAD methodology.

According to the creators of UML, any modern OO approach to developing IS must be (1) use-case driven, (2) architecture-centric, and (3) iterative and incremental.

Use-Case Driven

- means that use cases are the primary modeling tool to define the behavior of the system.
- A use case describes how the user interacts with the system to perform some activity,
  - such as placing an order, making a reservation, or searching for information.
- The use cases are used
  - to identify
  - to communicate

  the requirements for the system to the programmers who must write the system.

43

Architecture Centric

- means that the underlying software architecture of the evolving system specification drives the specification, construction, and documentation of the system.
- Modern OO systems analysis and design approaches should support at least three separate but interrelated architectural views of a system:
  - functional
  - static,
  - dynamic.
- Any modern approach to systems analysis and design should be architecture centric.

44

Iterative and Incremental

- Modern OO systems analysis and design approaches emphasize iterative and incremental development that
  - undergoes continuous testing and refinement throughout the life of the project.
- Each iteration of the system brings it closer and closer to real user needs.

45

The Unified Process

- The Unified Process is a specific methodology that maps out when and how to use the various UML techniques for OO analysis and design.
- Whereas the UML provides structural support for developing the structure and behavior of an information system, the Unified Process provides the behavioral support.
- The Unified Process is use-case driven, architecture centric, and iterative and incremental.

46

The Unified Process

- The Unified Process is a two-dimensional systems development process described by a set of phases and workflows. The phases are
  - inception, elaboration, construction, and transition.
- The workflows include
  - business modeling, requirements, analysis, design, implementation, test, deployment, project management, configuration and change management, and environment.

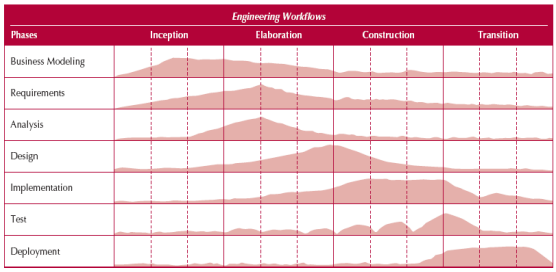
47

Phases

- The phases of the Unified Process support an analyst in developing information systems in an iterative and incremental manner.
- The phases describe how an information system evolves through time.
- Depending on which development phase the evolving system is currently in, the level of activity will vary over the workflows.
- The curves, in the next figures, associated with each workflow approximates the amount of activity that takes place during the specific phase.

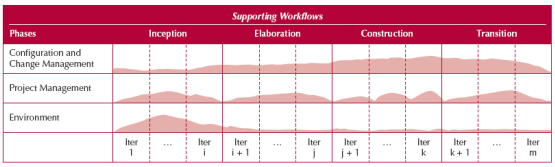
48

Engineering Workflows



49

Supporting Workflows



50

Phases

- Inception
- very similar to the planning phase of a traditional SDLC approach.
- A business case is made for the proposed system.
- This includes feasibility analysis that should answer questions such as the following:
  - Do we have the technical capability to build it? (technical feasibility)
  - If we build it, will it provide business value? (economic feasibility)
  - If we build it, will it be used by the organization? (organizational feasibility)

51

Phases

- To answer these questions,
  - the development team performs work related primarily to the business modeling, requirements, and analysis workflows.
- The project management and environment supporting workflows are very relevant to this phase.
- The primary deliverables from the inception phase are
  - a vision document that sets the scope of the project, identifies the primary requirements and constraints, sets up an initial project plan, and describes the feasibility of and risks associated with the project
  - the adoption of the necessary environment to develop the system.

52

Phases

- Elaboration
- The analysis and design workflows are the primary focus during this phase.
- The elaboration phase continues with developing the vision document, including
  - finalizing the business case,
  - revising the risk assessment,
  - completing a project plan in sufficient detail

  to allow the stakeholders to be able to agree with constructing the actual final system.
- It deals with gathering the requirements, building the UML structural and behavioral models of the problem domain, and detailing the how the problem domain models fit into the evolving system architecture.

53

Phases

- Developers are involved with all but the deployment engineering workflow in this phase.
- As the developers iterate over the workflows, the importance of addressing configuration and change management becomes apparent.
- The primary deliverables of this phase include
  - the UML structure and behavior diagrams
  - an executable of a baseline version of the evolving information system.
- By providing a solid foundation at this point in time, the developers can begin to grow the system toward its completion in the construction and transition phases.

54

Phases

- Construction
- focused on programming the evolving information system.
- primarily concerned with the implementation workflow.
- the requirements, analysis, and design workflows also are involved with this phase.
- It is during this phase that missing requirements are uncovered, and the analysis and design models are finally completed.

55

Phases

- There are iterations of the workflows during this phase.
- The configuration and change management workflow, with its version control activities, becomes extremely important during the construction phase.
- At times, an iteration may have to be rolled back.
  - Without good version controls, rolling back to a previous version (incremental implementation) of the system is nearly impossible.
- The primary deliverable of this phase
  - an implementation of the system that can be released for beta and acceptance testing.

56

Phases

- Transition
- addresses aspects associated with the implementation phase of a traditional SDLC approach.
- Its primary focus is on the testing and deployment workflows.
- The business modeling, requirements, and analysis workflows should have been completed in earlier iterations of the evolving information system.
- From a managerial perspective, the project management, configuration and change management, and environment are involved.

57

Phases

- Some of the activities that take place are
  - beta and acceptance testing,
  - fine tuning the design and implementation,
  - user training,
  - the actual rolling out of the final product onto a production platform.
- The primary deliverable is
  - the actual executable information system.
- The other deliverables include
  - user manuals, a plan to support the users, and a plan for upgrading the information system in the future.

58

Workflows

- describe the tasks or activities that
  - a developer performs to evolve an information system over time.
- The workflows of the Unified Process are grouped into two broad categories:
  - engineering workflows
  - supporting workflows

59

Engineering Workflows

- include
  - business modeling workflow,
  - requirements workflow,
  - analysis workflow,
  - design workflow,
  - implementation workflow,
  - test workflow,
  - deployment workflow.
- deal with the activities that produce the technical product
  - i.e., the Information System.

60

Business Modeling workflow

- uncovers problems and identifies potential projects within a user organization.
- aids management in understanding the scope of the projects that can improve the efficiency and effectiveness of a user organization.
- The primary purpose of business modeling is to ensure that both developer and user organizations understand where and how the to-be-developed information system fits into the business processes of the user organization.
- This workflow primarily is executed during the inception phase to ensure that we develop information systems that make business sense.
- The activities that take place on this workflow are most closely associated with the planning phase of the traditional SDLC

61

Requirements workflow

- includes eliciting both functional and nonfunctional requirements.
- Typically, requirements are gathered from project stakeholders, such as end users, managers within the end user organization, and even customers.
  - There are many different ways in which to capture requirements,
    - interviews, observation techniques, joint application development, document analysis, and questionnaires.
- utilized the most during the inception and elaboration phases.
- The identified requirements are very useful in developing the vision document and the use cases used throughout the development process.
- It should be stressed that additional requirements tend to be discovered throughout the development process.

62

Analysis workflow

- addresses creating an analysis model of the problem domain.
  - In the Unified Process, the analyst begins designing the architecture associated with the problem domain,
  - and using the UML, the analyst creates structural and behavioral diagrams that depict a description of the problem domain classes and their interactions.
- The primary purpose of the analysis workflow
  - to ensure that both the developer and user organizations understand the underlying problem and its domain without over analyzing.
- If they are not careful, analysts can create analysis paralysis,
  - occurs when the project becomes so bogged down with analysis that the system is never actually designed or implemented.

63

Analysis workflow

- A second purpose of the analysis workflow
  - to identify useful reusable classes for class libraries.
- By reusing predefined classes, the analyst can avoid "reinventing the wheel" when creating the structural and behavioral diagrams.
- The analysis workflow is predominantly associated with the elaboration phase,
  - but like the requirements workflow, it is possible that additional analysis will be required throughout the development process.

64

Design workflow

- transitions the analysis model into a form that can be used to implement the system:
  - the design model.
- focuses on developing a solution that will execute in a specific environment.
- enhances the evolving information system description by adding classes that address the environment of the information system to the evolving analysis model.
- As such, the design workflow addresses activities, such as
  - user interface design, database design, physical architecture design, detailed problem domain class design, and the optimization of the evolving information system.
- The design workflow primarily is associated with the elaboration and construction phases of the Unified Process.

65

Implementation workflow

- The primary purpose is to create an executable solution based on the design model including
  - writing new classes,
  - incorporating reusable classes from executable class libraries into the evolving solution.
- Testing of the new classes and their interactions with the incorporated reusable classes must occur.
- In the case of multiple groups performing the implementation of the information system,
  - the implementers also must integrate the separate, individually tested, modules to create an executable version of the system.
- Associated with the elaboration and construction phases.

66

Test workflow

- The primary purpose is to increase the quality of the evolving system.
- includes
  - testing the integration of all modules used to implement the system,
  - user acceptance testing,
  - the actual alpha testing of the software.
- testing of the analysis and design models are involved during the elaboration and construction phases,
- implementation testing is performed primarily during the construction and, to some degree, transition phases.
  - At the end of each iteration during the development of the information system, some type of test should be performed.

67

Deployment workflow

- most associated with the transition phase of the Unified Process.
- includes activities, such as
  - software packaging,
  - distribution,
  - installation,
  - beta testing.
- When actually deploying the new information system into a user organization,
  - the developers may have to convert the current data,
  - interface the new software with the existing software,
  - provide end user training on the use of the new system.

68

Supporting Workflows

- include
  - the project management workflow,
  - configuration and change management workflow,
  - the environment workflow.
- focus on the managerial aspects of information system development.

69

Project management workflow

- cross-phase workflow.
- This workflow's activities include
  - risk identification and management,
  - scope management,
  - estimating the time to complete each iteration and the entire project,
  - estimating the cost of the individual iteration and the whole project,
  - tracking the progress being made toward the final version of the evolving information system.

70

Configuration and change management workflow

- The primary purpose is to keep track of the state of the evolving system.
- The evolving information system comprises a set of artifacts that includesdiagrams, source code, and executables.
- During the development process, these artifacts are modified.
- Since the artifacts are modified on a regular, if not continuous, basis, good version control mechanisms should be established.
- A good deal of project management information needs to be captured
  - e.g., author, time, and location of each modification.
- associated mostly with the construction and transition phases.

71

Environment workflow

- During the development of an information system, the development team needs to use different tools and processes.
  - The environment workflow addresses these needs.
  - For example,
    - a computer aided software engineering tool that supports the development of an OO information system via the UML could be required.
- Other tools necessary would include
  - programming environments,
  - project management tools,
  - configuration management tools.
- The environment workflow includes acquiring and installing these tools.
- Environment workflow should primarily be involved with the inception phase.

72

A MINIMALIST APPROACH TO OO SYSTEMS ANALYSIS AND DESIGN WITH UML 2.0

The UML is an OO modeling language used to describe information systems.

It provides a common vocabulary of OO terms and a set of diagramming techniques that are rich enough to model any systems development project from analysis through implementation.

UML is nothing more than a notation.

UML does not dictate any formal approach to developing information systems, but its iterative nature is best-suited to RAD-based approaches such as phased development

A popular RAD-based approach that uses the UML is the Unified Process.

Benefits of Object-Oriented Systems Analysis and Design

- Concepts in the OO approach enable analysts to
  - break a complex system into smaller, more manageable modules,
  - work on the modules individually,
  - easily piece the modules back together to form an IS.
- This modularity makes system development
  - easier to grasp,
  - easier to share among members of a project team,
  - easier to communicate to users
    - who are needed to provide requirements and confirm how well the system meets the requirements throughout the SDLC.
- By modularizing system development, the project team actually is creating reusable pieces that can be plugged into other systems efforts, or used as starting points for other projects.

74

Benefits of Object-Oriented Systems Analysis and Design

- Ultimately, this can save time because new projects don't have to start completely from scratch.
- Finally, many people argue that "object-think" is a much more realistic way to think about the real world.
- Users typically do not think in terms of data or process;
  - instead, they see their business as a collection of logical units that contain both
    - so communicating in terms of objects improves the interaction between the user and the analyst or developer.
- Next figure summarizes the major concepts of the object-oriented approach and how each concept contributes to the benefits.

75

Benefits of Object-Oriented Systems Analysis and Design

Concept	Supports	Leads to
Classes, objects, methods, and messages	■ A more realistic way for people to think about their business ■ Highly cohesive units that contain both data and processes	■ Better communication between user and analyst or developer ■ Reusable objects ■ Benefits from having a highly cohesive system (see cohesion in Chapter 13)
Encapsulation and information hiding	■ Loosely coupled units	■ Reusable objects ■ Fewer ripple effects from changes within an object or in the system itself ■ Benefits from having a loosely coupled system design (see coupling in Chapter 13)
Inheritance	■ Allows us to use classes as standard templates from which other classes can be built	■ Less redundancy ■ Faster creation of new classes ■ Standards and consistency within and across development efforts ■ Ease in supporting exceptions
Polymorphism and Dynamic Binding	■ Minimal messaging that is interpreted by objects themselves	■ Simpler programming of events ■ Ease in replacing or changing objects in a system ■ Fewer ripple effects from changes within an object or in the system itself
Use-case driven and use cases	■ Allows users and analysts to focus on how a user will interact with the system to perform a single activity	■ Better understanding and gathering of user needs ■ Better communication between user and analyst
Architecture centric and functional, static, and dynamic views	■ Viewing the evolving system from multiple points of view	■ Better understanding and modeling of user needs ■ More complete depiction of information system
Iterative and incremental development	■ Continuous testing and refinement of the evolving system	■ Meeting real needs of users ■ Higher quality systems

76

Extensions to the Unified Process

- Unified Process some critical weaknesses.
- First, the UP does not address
  - staffing issues,
  - budgeting issues,
  - contract management issues.
- These activities were explicitly left out of the UP.
- Second, the UPdoes not address issues relating to
  - maintenance,
  - operations,
  - support of the product once it has been delivered.

77

Extensions to the Unified Process

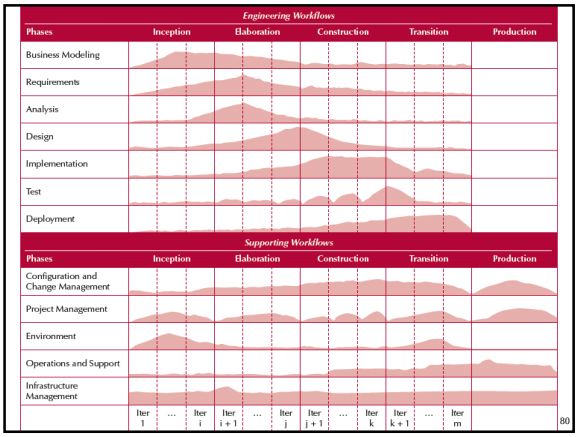
- Third, the Unified Process does not address cross- or inter- project issues.
- Considering the importance of reuse in OO systems development and the fact that in many organizations employees work on many different projects at the same time, leaving out inter-project issues is a major omission.

78

Extensions to the Unified Process

- To address these omissions, Ambler and Constantine suggest the addition of a Production phase and two workflows:
  - Operations and Support workflow
  - Infrastructure Management workflow
- In addition to these new workflows,
  - the test, deployment and environment workflows are modified, and the project management and configuration and change management workflows are extended into the production phase.

79



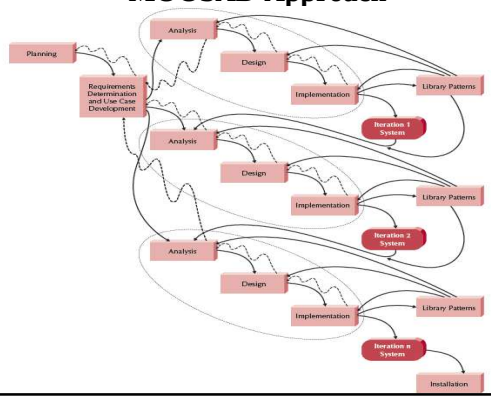
80

The Minimalist OO SAD Approach

- OOSAD approaches are based on a phased-development RAD approach.
- However, because of the iteration across the functional, static, and dynamic views of the evolving information system,
  - an actual OO development process tends to be more complex than typical phased-development RAD approaches.
- The minimalist OOSAD (MOOSAD) approach is based on the Unified Process
  - as extended by the processes associated with the OPEN Process and the OO Software Process approaches to object-oriented systems development.
- Next Figure shows modified phased-development RAD-based approach.
  - The solid lines in represent information flows from one step to another step.
  - The dashed lines represent feedback from a later step to an earlier one.

81

MOOSAD Approach



82

Summary

- Class and method design
- Data management layer design
- Human computer interaction layer design
- Physical architecture layer design
- Construction
- Installation
- Operations and support
- Basic characteristics of an object oriented system
- Unified modeling system
- Object oriented Systems Analysis and Design
- Minimalist approach to OO SAD with UML

83