Prof. Dr. Nizamettin AYDIN

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

Moving On To Design

The design phase in OO system development uses the requirements that were gathered during analysis to create a blueprint for the future system.

A successful design builds upon what was learned in earlier phases and leads to a smooth implementation by creating a clear, accurate plan of what needs to be done.

Key Ideas

- The purpose of the analysis phase is to figure out
  - what the business needs.
- The purpose of the design phase is to figure out
  - how to provide it.
- The steps in both analysis and design phases are highly interrelated
  - may require much “going back and forth”

Objectives

- Understand the transition from analysis to design.
- Understand the use of factoring, partitions, and layers.
- Be able to create package diagrams.
- Be familiar with the custom, packaged, and outsource design alternatives.
- Be able to create an alternative matrix.

Introduction

- An important initial part of the design phase
  - to examine several design strategies and decide which will be used to build the system.
- Systems can be
  - built from scratch,
  - purchased and customized,
  - outsourced to others.
- The project team needs to investigate the viability of each alternative.
  - The decision to make, to buy, or to outsource influences the design tasks that are accomplished throughout the rest of the phase.

Introduction

- Detailed design of the individual classes and methods that are used to map out the nuts and bolts of the system and how they are to be stored must still be completed.
- Techniques like
  - CRC cards,
  - class diagrams,
  - contract specification,
  - method specification,
  - database design

  provide detail in preparation for the implementation phase, and they ensure that programmers have sufficient information to build the right system efficiently.

Introduction

- Design also includes activities like designing
  - the user interface,
  - system inputs,
  - system outputs,

  which involve the ways that the user interacts with the system.
- Additionally, there are techniques such as
  - story boarding
  - prototyping

  that help the project team design a system that meets the needs of its users.

7

Introduction

- Finally, physical architecture decisions are made:
  - the hardware and software that will be purchased to support the new system and
  - the way that the processing of the system will be organized.
  - For example,
    - the system can be organized so that its processing is centralized at one location, distributed, or both centralized and distributed, and each solution offers unique benefits and challenges to the project team.
- Global issues and security need to be considered along with the system's technical architecture
  - because they will influence the implementation plans that are made.

8

Introduction

- Many steps of the design phase are highly interrelated
  - the analysts often go back and forth among them.
- For example,
  - prototyping in the interface design step often uncovers additional information that is needed in the system.
  - Alternatively, a system that is being designed for an organization that has centralized systems may require substantial hardware and software investments
    - if the project team decides to change to a system in which all of the processing is distributed.
- packages and package diagrams will be introduced
- three fundamental approaches (make, buy, outsource) to developing new systems will be examined

9

EVOLVING THE ANALYSIS MODELS INTO DESIGN MODELS

The purpose of the analysis models was
- to represent the underlying business problem domain as a set of collaborating objects.

the primary purpose of the design models is
- to increase the likelihood of successfully delivering a system that implements the functional requirements in a manner that is affordable and easily maintainable.

- Therefore, in system design, both the functional and nonfunctional requirements are addressed.
- From an OO perspective, system design models simply refine the system analysis models
  - by adding system environment (or solution domain) details to them
  - by refining the problem domain information already contained in the analysis models.

11

- When evolving the analysis model into the design model, the use cases and the current set of classes (their methods and attributes, and the relationships between them) should be reviewed.
  - Are all of the classes necessary?
  - Are there any missing classes?
  - Are the classes fully defined?
  - Are there any missing attributes or methods?
  - Do the classes have any unnecessary attributes and methods?
  - Is the current representation of the evolving system optimal?
- Next, factoring, partitions and collaborations, and layers as a way to evolve problem domain-oriented analysis models into optimal solution domain-oriented design models are introduced.

12

Avoid Classic Design Mistakes

- **Reducing design time**
- If time is short, there is a temptation to reduce the time spent in “unproductive” activities such as design so that the team can jump into “productive” programming.
- This results in missing important details that have to be investigated later at a much higher time cost
  - usually at least ten times longer.
- **Solution:**
- If time pressure is intense, use timeboxing to eliminate functionality or move it into future versions.

Avoid Classic Design Mistakes

- **Feature creep**
- Even if you are successful at avoiding scope creep, about 25 percent of system requirements will still change.
  - Changes can significantly increase time and cost.
- **Solution:**
- Ensure that all changes are vital and that the users are aware of the impact on cost and time.
- Try to move proposed changes into future versions.

Avoid Classic Design Mistakes

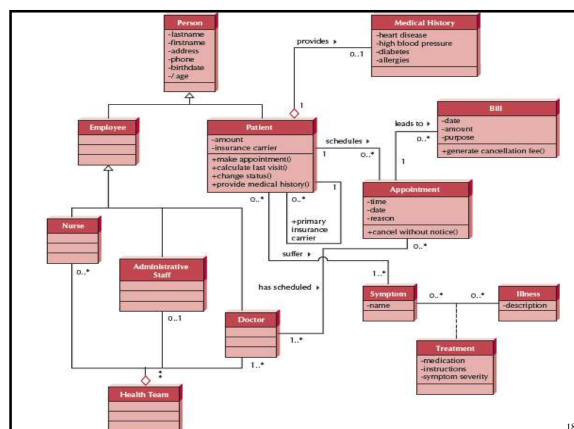
- **Silver bullet syndrome**
- Analysts sometimes believe the marketing claims for some design tools that claim to solve all problems and magically reduce time and costs.
  - No one tool or technique can eliminate overall time or costs by more than 25%).
- **Solution:**
- If a design tool has claims that appear too good to be true,
  - just say no.

Avoid Classic Design Mistakes

- **Switching tools in mid-project**
- Sometimes analysts switch to what appears to be a better tool during design in the hopes of saving time or costs.
  - Usually, any benefits are outweighed by the need to learn the new tool.
  - This also applies to even “minor” upgrades to current tools.
- **Solution:**
- Don’t switch or upgrade unless there is a compelling need for specific features in the new tool, and then explicitly increase the schedule to include learning time.

Factoring

- the process of separating out a module into a standalone module in and of itself.
  - The new module can be a new class or a new method.
  - For example,
  - when reviewing a set of classes, it may be discovered that they have a similar set of attributes and methods.
    - As such, it may make sense to factor out the similarities into a separate class.
  - Depending on whether the new class should be in a superclass relationship to the existing classes or not, the new class can be related to the existing classes through generalization (A-Kind-Of) or possibly through aggregation (Has-Parts) relationship.



Factoring

- For example, using the appointment system example discussed previously.
- if the Employee class had not been identified, it could possibly be identified at this stage
- by factoring out the similar methods and attributes from the Nurse, Administrative Staff, and Doctor classes.
- In this case, we would relate the new class (Employee) to the existing classes using the generalization (A-Kind-Of) relationship.

19

Factoring

- Abstraction and refinement are two closely related processes to factoring.
- Abstraction
  - deals with the creation of a "higher" level idea from a set of ideas.
    - Identifying the Employee class is an example of abstracting from a set of lower classes to a higher one.
  - In some cases, the abstraction process will identify abstract classes
  - In other situations, it will identify additional concrete classes.

20

Factoring

- The refinement process is the opposite of the abstraction process.
- In the appointment system example (in the previous eample), it is possible to identify additional subclasses of the Administrative Staff class, such as Receptionist, Secretary, and Bookkeeper.
- We would only add the new classes if there were sufficient differences between them.
- Otherwise, the more general class, Administrative Staff, would suffice.

21

Partitions and Collaborations

- the sheer size of the system representation can overload both the user and the developer.
- At this point in the evolution of the system, it may make sense to split the representation into a set of partitions.
- A partition
  - the OO equivalent of a subsystem,
    - where a subsystem is a decomposition of a larger system into its component systems
    - (e.g., an accounting information system could be functionally decomposed into an accounts payable system, an account receivable system, a payroll system, and so on).
  - From an OO perspective, partitions are based on the pattern of activity (messages sent) among the objects in an OO system.

22

Partitions and Collaborations

- A good place to look for potential partitions is the collaborations modeled in UML's communication diagrams
- CRUD analysis can also be used to identify potential classes on which to merge collaborations.
- Depending on the complexity of the merged collaboration, it may be useful in decomposing the collaboration into multiple partitions.
  - In this case, in addition to having collaborations between objects, it is possible to have collaborations among partitions.

23

Partitions and Collaborations

- Another useful approach to identify potential partitions is to model each collaboration between objects in terms of clients, servers, and contracts.
- A client
  - an instance of a class that sends a message to an instance of another class for a method to be executed
- A server
  - the instance of a class that receives the message
- A contract
  - the specification that formalizes the interactions between the client and server objects

24

Partitions and Collaborations

- This approach allows the developer to build up potential partitions by looking at the contracts that have been specified between objects.
- In this case,
  - the more contracts there are between objects,
  - the more often than not the objects belong in the same partition.
- The fewer contracts,
  - the chances are the two classes do not belong in the same partition.

25

Layers

- Until this point in the development of our system, we have focused only on the problem domain.
- To successfully evolve the analysis model of the system into a design model of the system,
  - we must add the system environment information.
- One useful way to do this, without overloading the developer, is to use layers.

26

Layers

- A layer
  - represents an element of the software architecture of the evolving system.
- We have focused only on one layer in the evolving software architecture:
  - the problem domain layer.
- There should be a layer for each of the different elements of the system environment
  - system architecture,
  - user interface,
  - data access and management.

27

Layers

- The idea of separating the different elements of the architecture into separate layers is traced back to the MVC architecture of *Smalltalk*.
- When Smalltalk was first created, the authors decided to separate the application logic from the logic of the user interface.
- In this manner, it was possible to easily develop different user interfaces that worked with the same application.

28

Layers

- To accomplish this, they created the Model-View-Controller (MVC) architecture where
  - Models implemented the application logic (problem domain),
  - Views and Controllers implemented the logic for the user interface.
    - Views handled the output
    - Controllers handled the input.
- Since graphical user interfaces were first developed in the Smalltalk language, the MVC architecture served as the foundation for virtually all graphical user interfaces that have been developed today
    - including the Mac interfaces, the Windows family, and the various Unix-based GUI environments.

29

Layers

- Based on Smalltalk's innovative MVC architecture, many different software layers have been proposed.
- Based on these proposals, the following layers on which to base software architecture may be suggested:
  - foundation,
  - physical architecture,
  - human computer interaction,
  - data access and management,
  - problem domain.
- Each layer limits the types of classes that can exist on it
  - e.g., only user interface classes may exist on the human computer interaction layer.

30

Foundation layer

- Contains classes that are necessary for any OO application to exist.
- They include
  - classes that represent fundamental data types
    - e.g., integers, real numbers, characters, and strings,
  - classes that represent fundamental data structures (sometimes referred to as container classes;
    - e.g., lists, trees, graphs, sets, stacks, and queues,
  - classes that represent useful abstractions (sometimes referred to as utility classes;
    - e.g., date, time, and money.
- Today, the classes found on this layer typically are included with the OO development environments.

31

Physical architecture layer

- addresses how the software will execute on specific computers and networks.
- includes classes that deal with communication between the software and the computer's operating system and the network.
  - For example, classes that address how to interact with the various ports on a specific computer would be included in this layer.
- also includes classes that would interact with so-called middleware applications, such as the OMG's CORBA and Microsoft's DCOM architectures that handle distributed objects.

32

- Object Management Group (http://www.omg.org)
  - OMG has been an international, open membership, not-for-profit computer industry consortium since 1989
    - OMG's mission is to develop, with our worldwide membership, enterprise integration standards that provide real-world value. OMG is also dedicated to bringing together end-users, government agencies, universities and research institutions in our communities of practice to share experiences in transitioning to new management and technology approaches like Cloud Computing.
- OMG's middleware standards and profiles are based on the Common Object Request Broker Architecture (CORBA®) and support a wide variety of industries.
  - http://www.corba.org/

33

Physical architecture layer

- There are many design issues that must be addressed before choosing the appropriate set of classes for this layer:
- The choice of
  - a computing or network architecture
    - such as the various client-server architectures,
  - the actual design of a network, hardware and server software specification,
  - global/international issues
    - such as multilingual requirements,
  - security issues.

34

Human computer interaction layer

- contains classes associated with the View and Controller idea from Smalltalk.
- The primary purpose of this layer is to keep the specific user interface implementation separate from the problem domain classes.
- This increases the portability of the evolving system.
- Typical classes found on this layer include classes that can be used to represent
  - buttons, windows, text fields, scroll bars, check boxes, drop-down lists, and many other classes that represent user interface elements.

35

Data management layer

- addresses the issues involving the persistence of the objects contained in the system.
- The types of classes that appear in this layer deal with how objects can be stored and retrieved.
- The classes contained in this layer allow the problem domain classes to be independent of the storage utilized, and hence increase the portability of the evolving system.
- Some of the issues related to this layer include choice of the storage format (such as relational, object/relational, and object databases) and storage optimization (such as clustering and indexing).

36

Problem domain layer

- what we have focused our attention on up until now.
- At this stage of the development of our system, we will need to further detail the classes so that it will be possible to implement them in an effective and efficient manner.
- Many issues need to be addressed when designing classes. For example,
  - there are issues related to
    - factoring, cohesion and coupling, connascence, encapsulation, proper use of inheritance and polymorphism, constraints, contract specification, and detailed method design.

37

PACKAGES AND PACKAGE DIAGRAMS

In UML, collaborations, partitions, and layers can be represented by a higher-level construct:
- a package.

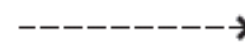
A package
- a general construct that can be applied to any of the elements in UML models.

A package diagram
- a class diagram that only shows packages.

Syntax for Package Diagram

- **A Package:**
- A logical grouping of UML elements
- Used to simplify UML diagrams
  - by grouping related elements into a single higher level element

- **A Dependency Relationship:**



- Represents a dependency between packages,
  - i.e., if a package is changed, the dependent package also could have to be modified
- The arrow is drawn from the dependent package toward the package on which it is dependent

39

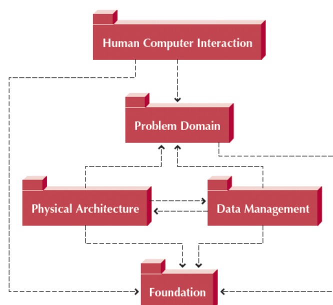
Package Diagram

- Depending on where a package is used,
  - packages can participate in different types of relationships.
- For example,
  - in a class diagram, packages represent groupings of classes.
  - Therefore, aggregation and association relationships are possible.
- A dependency relationship represents the fact that a modification dependency exists between two packages.
  - It is possible that a change in one package potentially could cause a change to be required in another package.
    - Following figure portrays the dependencies among the different layers
      - foundation, physical architecture, human computer interaction, data access and management, and problem domain.

40

Package Diagram of Dependency Relationships among Layers

- If a change occurs in the problem domain layer,
  - it most likely will cause changes to occur in the
    - human computer interaction,
    - physical architecture,
    - data management layers.
- Notice that these layers "point to" the problem domain layer;
  - as such, they are dependent on it.
- However,
  - the reverse is not true.



41

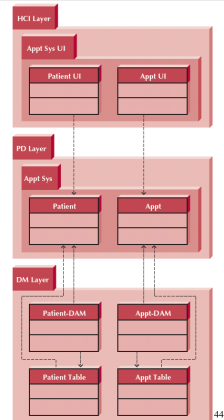
Modification Dependency

- Indicates that a change in one package could cause a change to be required in another package.
  - At the class level, there could be many causes for dependencies among classes.
- For example,
  - A change in one method will cause the interface for all objects of this class to change.
  - Therefore, all classes that have objects that send messages to the instances of the modified class may have to be modified.
- Capturing dependency relationships among the classes and packages helps the organization in maintaining OO information systems.

42

- Collaborations, partitions, and layers are modeled as packages in UML.
  - collaborations are normally factored into a set of partitions, which are typically placed on a layer.
  - In addition, partitions can be composed of other partitions.
  - Also, it is possible to have classes in partitions, which are contained in another partition, which is placed on a layer.
- All of these groupings are represented using packages in UML.
  - a package is simply a generic, grouping construct used to simplify UML models through the use of composition.

- A simple package diagram, based on the appointment system example from the previous slides, is shown in this figure.
  - This diagram portrays a very small portion of the entire system.
- we see that the Patient UI, DAM-Patient, and Patient Table classes are dependent on the Patient class.
- Furthermore, the DAM-Patient class is dependent on the Patient Table class.
  - The same can be seen with the classes dealing with the actual appointments.
- By isolating the Problem Domain classes (such as the Patient and Appt classes) from the actual object persistence classes (such as the Patient Table and Appt Table classes) through the use of the intermediate Data Access and Manipulation classes (DAM-Patient and DAM-Appt classes), we isolate the Problem Domain classes from the actual storage medium.
- This greatly simplifies the maintenance and increases the reusability of the Problem Domain classes.
- Of course, in a complete description of a real system, there would be many more dependencies.



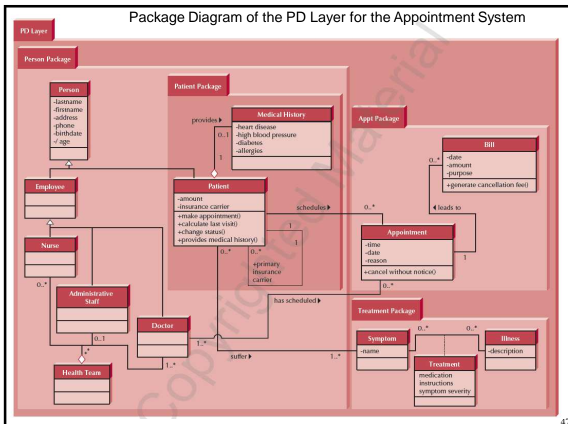
Identifying Packages and Creating Package Diagrams

- set the context for the package diagram.
  - Remember, packages can be used to model partitions and/or layers.
    - Revisiting the appointment system again, let's set the context as the problem domain layer.
- cluster the classes together into partitions based on the relationships that the classes share.
  - The relationships include generalization, aggregation, the various associations, and the message sending that takes place between the objects in the system.
    - To identify the packages in the appointment system, we should look at the different analysis models (e.g., the class diagram, sequence diagrams, and the communication diagrams).
    - Any classes in a generalization hierarchy should be kept together in a single partition.

Identifying Packages and Creating Package Diagrams

- place the clustered classes together in a partition and model the partitions as packages.
- Following figure portrays five packages:
  - PD Layer,
  - Person Pkg,
  - Patient Pkg,
  - Appt Pkg,
  - Treatment Pkg.

Package Diagram of the PD Layer for the Appointment System



Copyrighted Material

Identifying Packages and Creating Package Diagrams

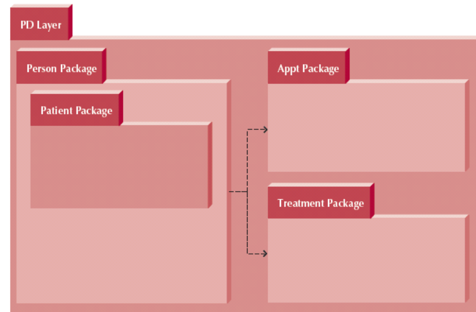
- identify the dependency relationships among the packages.
  - In this case, we review the relationships that cross the boundaries of the packages to uncover potential dependencies.
  - In the appointment system, we see association relationships
    - connecting the Person Pkg with the Appt Pkg (via the association between the Doctor class and the Appointment class), and the Patient Pkg, which is contained within the Person Pkg, with the Appt Pkg (via the association between the Patient and Appointment classes) and the Treatment Pkg (via the association between the Patient and Symptom classes).

Identifying Packages and Creating Package Diagrams

- place the dependency relationships on the evolved package diagram.
  - In the case of the Appointment system,
    - there are dependency relationships between the Person Pkg and the Appt Pkg and the Person Pkg and the Treatment Pkg.
  - To increase the understandability of the dependency relationships among the different packages,
    - a pure package diagram that only shows the dependency relationships among the packages can be created
      - as shown in the following figure.

49

Overview Package Diagram of the PD Layer for the Appointment System



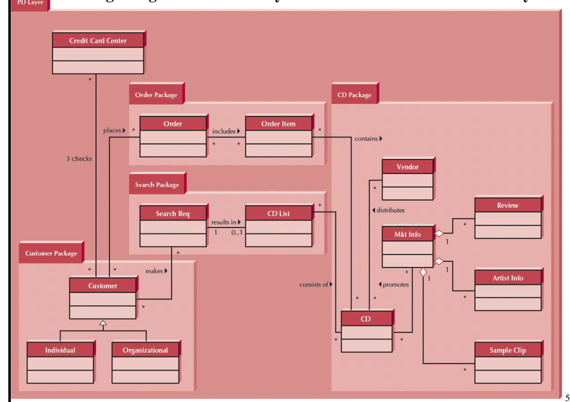
50

Example: CD Selections

- Creation of a package diagram for the CD Selections Internet sales system:
  - set the context
  - cluster classes together
  - model each partitions as packages
  - identify dependency relationships among the packages
  - place the dependency relationships on the package diagram

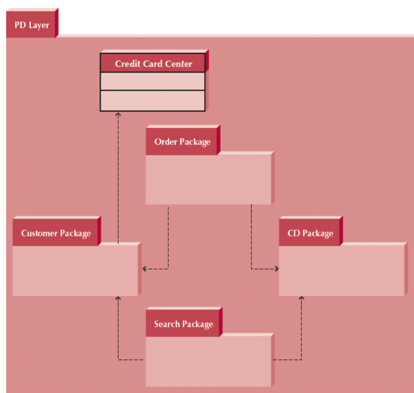
51

Package Diagram of the PD Layer of CD Selections Internet Sales System



52

Overview Package Diagram of the PD Layer of CD SelectionsInternet Sales System



53

DESIGN STRATEGIES

There are three ways to approach the creation of a new system:

- developing a custom application in-house,
- buying a packaged system and customizing it,
- relying on an external vendor, developer, or service provider to build the system.

Each of these choices has its strengths and weaknesses, and each is more appropriate in different scenarios.

Custom Development

- Many project teams assume that custom development,
  - building a new system from scratch,

  is the best way to create a system.
- Teams have complete control over the way the system looks and functions.

- custom development
  - allows for meeting highly specialized requirements
  - allows flexibility and creativity in solving problems

55

Custom Development

- Building a system in-house also builds technical skills and functional knowledge within the company.
- As developers work with business users, their understanding of the business grows and they become better able to align IS with strategies and needs.
- These same developers climb the technology learning curve so that future projects applying similar technology become much less effortful.

56

Custom Development

- However,
  - Custom application development,
    - includes a dedicated effort that requires long hours and hard work.
  - Many companies have a development staff that already is overcommitted to filling huge backlogs of systems requests, and they just do not have time for another project.
  - Also, a variety of skills—
    - technical, interpersonal, functional, project management, and modeling

    all have to be in place for the project to move ahead smoothly.
  - IS professionals, especially highly skilled individuals, are quite difficult to hire and retain.

57

Custom Development

- The risks associated with building a system from the ground up can be quite high,

- There is no guarantee that the project will succeed.

  - Developers could be pulled away to work on other projects,

  - Technical obstacles could cause unexpected delays,

  - The business users could become impatient with a growing timeline.

58

Packaged Software

- Many business needs are not unique, therefore
  - many organizations buy packaged software
    - Software that already been written.
- Most companies have needs that can be met quite well by packaged software
  - such as payroll or accounts receivable.
- It can be much more efficient to buy programs that have already been created, tested, and proven,
- A packaged system can be bought and installed in a relatively short period of time, when compared with a custom system.
- Packaged systems also incorporate the expertise and experience of the vendor who created the software.

59

Packaged Software

- Packaged software can range from reusable components
  - such as ActiveX and Javabeans
- to small single-function tools
  - like the shopping cart program
- to huge, all encompassing systems
  - such as enterprise resource planning (ERP) applications that are installed to automate an entire business.
    - Implementing ERP systems is a process in which large organizations spend millions of dollars installing packages by companies like
      - SAP, PeopleSoft, Oracle, and Baan

      and then change their businesses accordingly.
  - Installing ERP software is much more difficult than installing small application packages
    - because benefits can be harder to realize and problems are much more serious.

60

Packaged Software

- One problem is that companies buying packaged systems must accept the functionality that is provided by the system.
- If the packaged system is large in scope,
  - its implementation could mean a substantial change in the way the company does business.
- Letting technology drive the business can be a dangerous way to go.
- Most packaged applications allow for customization,
  - the manipulation of system parameters

  to change the way certain features work.

61

Packaged Software

- If the amount of customization is not enough,
- If the software package has a few features that don't quite work the way the company needs it to work,
  - the project team can create workarounds.
- A workaround
  - a custom-built add-on program
    - interfacing with the packaged application to handle special needs.
- It can be a nice way to create needed functionality that does not exist in the software package.
- But, workarounds should be a last resort for several reasons:

62

Packaged Software

- First, workarounds are not supported by the vendor who supplied the packaged software,
  - so when upgrades are made to the main system, they may make the workaround ineffective.
- Also, if problems arise, vendors have a tendency to blame the workaround as the culprit and refuse to provide support.
- Although choosing a packaged software system is simpler than custom development,
  - it too can benefit from following a formal methodology just as if you were building a custom application.

63

Packaged Software

- Systems integration
- refers to the process of building new systems by combining
  - packaged software,
  - existing legacy systems,
  - new software written to integrate these together.
- Many consulting firms specialize in systems integration,
  - so it is common for companies to select the packaged software option and then outsource the integration of a variety of packages to a consulting firm.

64

Packaged Software

- Systems integration
- refers to the process of building new systems by combining
  - packaged software,
  - existing legacy systems,
  - new software written to integrate these together.
- Many consulting firms specialize in systems integration,
  - so it is common for companies to select the packaged software option and then outsource the integration of a variety of packages to a consulting firm.

65

Packaged Software

- The key challenge in systems integration
  - finding ways to integrate the data produced by the different packages and legacy systems.
    - Integration often hinges around taking data produced by one package/system and reformatting it for use in another package/system.
  - The project team starts by examining the data produced by and needed by the different packages/systems and identifying the transformations that must occur to move the data from one to the other.
    - In many cases, this involves "fooling" the different packages/systems into thinking that the data was produced by an existing program module that the package/system expects to produce the data rather than the new package/system that is being integrated.

66

Packaged Software

- Another approach is through the use of an object wrapper.
- An object wrapper
  - an object that "wraps around" a legacy system,
    - enabling an object-oriented system to send messages to the legacy system.
- Object wrappers create an application program interface (API) to the legacy system.
- The creation of an object wrapper allows the protection of the corporation's investment in the legacy system.

67

Outsourcing

- The design choice that requires the least amount of in-house resources
- Outsourcing
- hiring an external vendor, developer, or service provider to create the system.
- There can be great benefit to having someone else develop your system:
  - They may be more experienced in the technology or have more resources, like experienced programmers.
  - Many companies embark upon outsourcing deals to reduce costs, while others see it as an opportunity to add value to the business.

68

Outsourcing

- outsourcing can be a good alternative for a new system.
- However, it does not come without costs.
  - If you decide to leave the creation of a new system in the hands of someone else,
    - you could compromise confidential information or lose control over future development.
  - In-house professionals are not benefiting from the skills that could be learned from the project,
    - instead the expertise is transferred to the outside organization.
  - Ultimately, important skills can walk right out the door at the end of the contract.

69

Outsourcing

- Most risks can be addressed if you decide to outsource, but two are particularly important:

1. assess the requirements for the project thoroughly
   - you should never outsource what you don't understand.
   - If you have conducted rigorous planning and analysis, then you should be well aware of your needs.
2. carefully choose a vendor, developer, or service with a proven track record with the type of system and technology that your system needs.

70

Outsourcing

- Three primary types of contracts can be drawn to control the outsourcing *contract.*
- A time and arrangements deal
  - very flexible because you agree to pay for whatever time and expenses are needed to get the job done.
    - Of course, this agreement could result in a large bill that exceeds initial estimates.
    - This works best when you and the outsourcer are unclear about what it is going to take to finish the job.
- You will pay no more than expected with a fixed-price contract
  - because if the outsourcer exceeds the agreed-upon price, they will have to absorb the costs.
  - Outsourcers are much more careful about defining requirements clearly up front, and there is little flexibility for change.

71

Outsourcing

- The type of contract gaining in popularity is the value-added contract
  - whereby the outsourcer reaps some percentage of the completed system's benefits.
    - You have very little risk in this case, but expect to share the wealth once the system is in place.
- Creating fair contracts is an art because you need to carefully balance flexibility with clearly defined terms.
  - Often needs change over time.
    - you don't want the contract to be so specific and rigid that alterations can't be made.
- Managing the outsourcing relationship is a full-time job.
  - Thus, someone needs to be assigned full-time to manage the outsourcer, and the level of that person should be appropriate for the size of the job

72

Outsourcing Guidelines

- Keep lines of communication open with outsourcer
- Define and stabilize requirements before signing a contract
- View outsourcing relationship as partnership
- Select outsource vender carefully
- Assign person to manage relationship
- Don't outsource what you don't understand
- Emphasize flexible requirements, long-term relationships, and short-term contracts

Selecting a Design Strategy

	Use Custom Development When…	Use a Packaged System When…	Use Outsourcing When…
Business Need	The business need is unique.	The business need is common.	The business need is not core to the business.
In-house Experience	In-house functional and technical experience exists.	In-house functional experience exists.	In-house functional or technical experience does not exist.
Project Skills	There is a desire to build in-house skills.	The skills are not strategic.	The decision to outsource is a strategic decision.
Project Management	The project has a highly skilled project manager and a proven methodology.	The project has a project manager who can coordinate vendor's efforts.	The project has a highly skilled project manager at the level of the organization that matches the scope of the outsourcing deal
Timeframe	The timeframe is flexible.	The timeframe is short.	The timeframe is short or flexible.

Your Turn

- Suppose that your university is interested in creating a new course registration system that can support Web-based registration?
- What should the university consider when determining whether to invest in a custom, packaged, or outsourcing system solution?

DEVELOPING THE ACTUAL DESIGN

Once the project team has a good understanding of how well each design strategy fits with the project's needs,

- they must begin to understand exactly how to implement these strategies

- For example,
  - what tools and technology would be used
    - if a custom alternative were selected?
  - What vendors make packaged systems that address the project needs?
  - What service providers would be able to build this system
    - if the application were outsourced?
- This information can be obtained
  - from people working in the IS department
  - from recommendations by business users.

- The project team can contact other companies with similar needs and investigate the types of systems that they have put in place.
- Vendors and consultants usually are willing to provide information about various tools and solutions in the form of brochures, product demonstrations, and information seminars.
  - However, be sure to validate the information you receive from vendors and consultants.
    - After all, they are trying to make a sale.
    - Therefore, they may "stretch" the capabilities of their tool by only focusing on the positive aspects of the tool while omitting the tool's drawbacks.

- It is likely that the project team will identify several ways that the system could be constructed after weighing the specific design options.
- For example,
  - the project team may have found three vendors that make packaged systems that potentially could meet the project's needs.
  - Or, the team may be debating over whether
    - to develop a system using Java as a development tool and the database management system from Oracle; or
    - to outsource the development effort to a consulting firm
  - Each alternative will have pros and cons associated with it that need to be considered, and only one solution can be selected in the end.

79

- An alternative matrix can be used to organize the pros and cons of the design alternatives
  - so that the best solution will be chosen in the end.
- This matrix is created using the same steps as the feasibility analysis.
  - The only difference is that the alternative matrix combines several feasibility analyses into one matrix
    - so that the alternatives can be easily compared.

80

The Alternative matrix

- a grid that contains
  - technical feasibilities,
  - budget feasibilities,
  - organizational feasibilities

    for each system candidate,
  - pros and cons associated with adopting each solution,
  - other information that is helpful when making comparisons.
- Sometimes weights are provided for different parts of the matrix
  - to show when some criteria are more important to the final decision.

81

The Alternative matrix

- To create the alternative matrix,
- draw a grid with
  - the alternatives across the top
  - different criteria
    - e.g., feasibilities, pros, cons, and other miscellaneous criteria)

    along the side.
- Next, fill in the grid with detailed descriptions about each alternative.
- This becomes a useful document for discussion
  - because it clearly presents the alternatives being reviewed and comparable characteristics for each one.

82

The Alternative matrix

- One helpful tool is the request for proposals (RFP).
- An RFP
  - a document that solicits proposals to provide the alternative solutions from a vendor, developer, or service provider.
  - explains the system that you are trying to build and the criteria that you will use to select a system.
- Vendors then respond by describing what it would mean for them to be a part of the solution.
- They communicate the time, cost, and exactly how their product or services will address the needs of the project.

83

The Alternative matrix

- an RFP should include basic information
  - the description of the desired system,
  - any special technical needs or circumstances,
  - evaluation criteria,
  - instructions for how to respond,
  - the desired schedule.
- An RFP can be
  - a very large document (i.e., hundreds of pages)
    - because companies try to include as much detail as possible about their needs so that the respondent can be just as detailed in the solution that would be provided.
  - Thus, RFPs typically are used for large projects rather than small ones
    - because they take a lot of time to create, and even more time and effort for vendors, developers, and service providers to develop high quality responses
      - only a project with a fairly large price tag would be worth the time and cost to develop a response for the RFP.

84

The Alternative matrix

- A less effort-intensive tool is
  - a request for information (RFI)
    - that includes the same format as the RFP.
- The RFI
  - shorter and contains less detailed information about a company's needs,
  - requires general information from respondents
    - that communicates the basic services that they can provide.

- The final step,
  - to decide which solution to design and implement.
- The decision should be made by a combination of
  - business users
  - technical professionals

  after the issues involved with the different alternatives are well understood.
- Once the decision is finalized,
  - the design phase can continue as needed,
    - based on the selected alternative.

Example: Applying the Concepts at CD Selections

- Three different approaches could be selected for the new system:
  - developing the entire system using development resources from CD Selections,
  - buying a commercial Internet sales packaged software program (or a set of different packages and integrate them),
  - hiring a consulting firm or service provider to create the system.
- Following slide shows alternative matrix for Shopping Cart Program

Alternative Matrix for Shopping Cart Program

	Alternative 1: Shop-With-Me	Alternative 2: WebShop	Alternative 3: Shop-N-Go
Technical Feasibility	• Developed using C: very little C experience in-house • Orders sent to company using email files	• Developed using C and JAVA: would like to develop in-house JAVA skills • Flexible export features for passing order information to other systems	• Developed using JAVA: would lik[e] to develop in-house JAVA skills • Orders saved to a number of file formats
Economic Feasibility	• $150 initial charge	• $700 up front charge, no yearly fees	• $200/year
Organizational Feasibility	• Program used by other retail music companies	• Program used by other retail music companies	• Brand new application: few companies have experience with Shop-N-Go to date
Other Benefits	• Very simple to use	• Tom in IS support has had limited, but positive experience with this program • Easy to customize	
Other Limitations			• The interface is not easily customized

Summary

- When evolving analysis into design models,
  - it is important to review the analysis models
  - then add system environment information.
- Packages and package diagrams help provide structure and less complex views of the new system.
- Custom building, packages, and outsourcing are alternative ways of creating the new system.
- The alternative matrix can help with the selection of a design strategy.

Expanding the Domain

- Smalltalk is an object-oriented programming language with many very loyal adherents.
- For more information check the site at:

  http://www.smalltalk.org/main.html