Object-Oriented Systems Development Life Cycle

Chapter 3

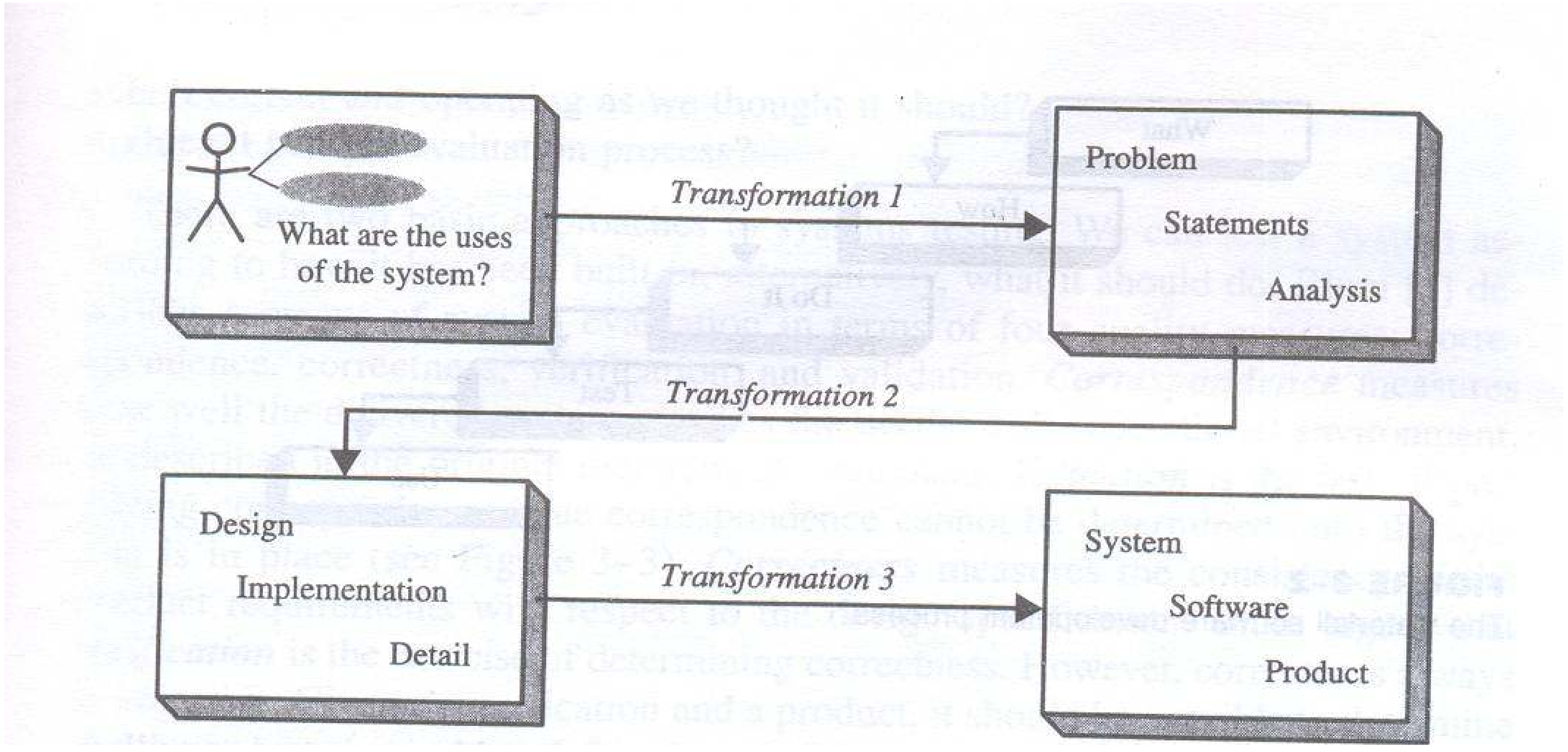
Introduction

- The essence of the software development process that consists of analysis, design, implementation, testing, and refinement is to transform user's needs into a software solution that satisfies those needs.
- Some people view software development process as interesting but feel it has little importance in developing software.

Introduction

- In general, dynamics of software development provides little room for such shortcuts, and bypasses have been less than successful.
- The object oriented approach requires a more rigorous process to do things right.
- You need not see code until after about 25 percent of the development time, because you need to spend more time in gathering requirements, developing a requirement model and an analysis model, then turning them into the design model.

The software development process



The software development process

- The software development process can be divided into smaller, interacting sub processes.
- Generally software development can be seen as a series of transformations, where the output of one transformation becomes the input of the subsequent transformation.

The software development process

- Transformation1 (Analysis):
  - Translates the user's needs into system requirements and responsibilities.
  - The way they use the system can provide insight into the user's requirements.
  - For example: one use of system might be analyzing an incentive payroll system, which will tell us that this capacity will be included in the system requirements.

The software development process

- Transformation 2 (design):
  - Begins with a problem statement and ends with a detailed design that can be transformed into a operational system.
  - This transformation includes the bulk of the software development activity, including the definition of how to build the software, its development, and its testing.
  - It also includes design descriptions, the programs and the testing material.

The software development process

- Transformation 3(Implementation)
  - refines the detailed design into the system deployment that will satisfy the user’s needs.
  - This takes into account equipment, procedures, people, and the like.
  - It represents embedding the software product within its operational environment.

Waterfall Model

- An example of the software development process is the waterfall approach which starts with deciding what has to be done.
- Once the requirements have been determined, we next must decide how to accomplish them.
- This is followed by a step in which we do it, whatever it has required us to do.
- We then must test the result to see if we have satisfied the user’s requirements.
- Finally we use the what we have done.

Waterfall Model

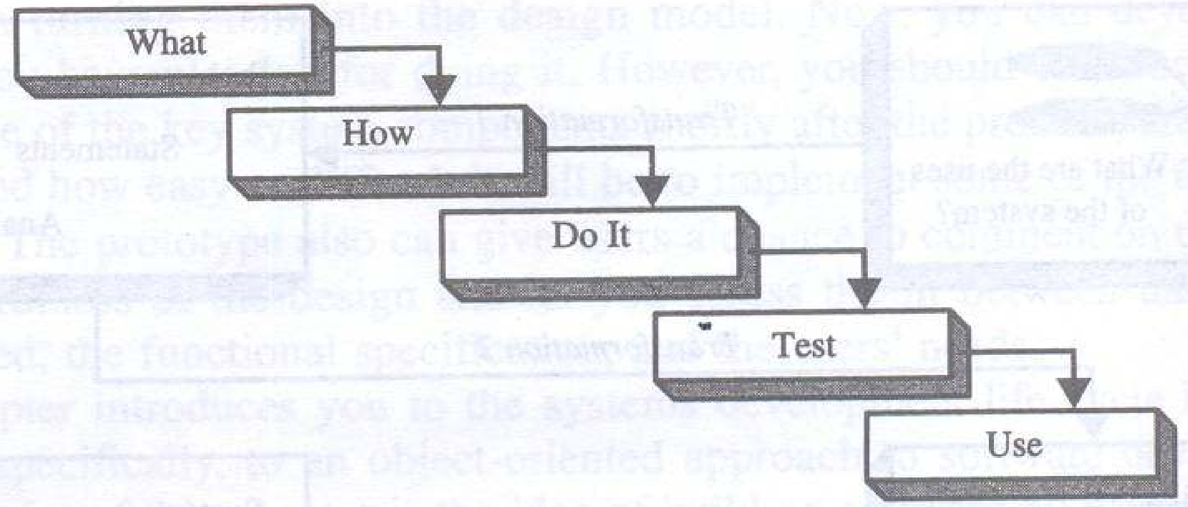


FIGURE 3–2
The waterfall software development process.

BUILDING HIGH-QUALITY SOFTWARE

- The software process transforms the users' needs via the application domain to a software solution that satisfies those needs.
- High-quality products must meet users' needs and expectations.
- Furthermore, the products should attain this with minimal or no defects, the focus being on improving products (or services) prior to delivery rather than correcting them after delivery.

BUILDING HIGH-QUALITY SOFTWARE

- The ultimate goal of building high-quality software is user satisfaction.
- To achieve high quality in software we need to be to answer the following questions:
  - • How do we determine when the system is ready for delivery?
  - • Is it now an operational system that satisfies users' needs?
  - • Is it correct and operating as we thought it should?
  - • Does it pass an evaluation process?

BUILDING HIGH-QUALITY SOFTWARE

- There are two basic approaches to systems testing.
- We can test a system according to how it has been built or, alternatively, what it should do.
- Blum describes a means of system evaluation in terms of four quality measures: correspondence, correctness, verification, and validation.

BUILDING HIGH-QUALITY SOFTWARE

- ***Correspondence*** measures how well the delivered system matches the needs of the operational environment, as described in the original requirements statement.
- ***Validation*** is the task of predicting correspondence. True correspondence cannot be determined until the system is in place.

BUILDING HIGH-QUALITY SOFTWARE

- ***Correctness*** measures the consistency of the product requirements with respect to the design specification.
- ***Verification*** is the exercise of determining correctness. However, correctness always is objective.
- Validation, however, is always subjective, and it addresses a different issue— the appropriateness of the specification.

BUILDING HIGH-QUALITY SOFTWARE

- Boehm observes that these quality measures, verification and validation, answer the following questions:
- Verification. Am I building the product right?
- Validation. Am I building the right product?

BUILDING HIGH-QUALITY SOFTWARE

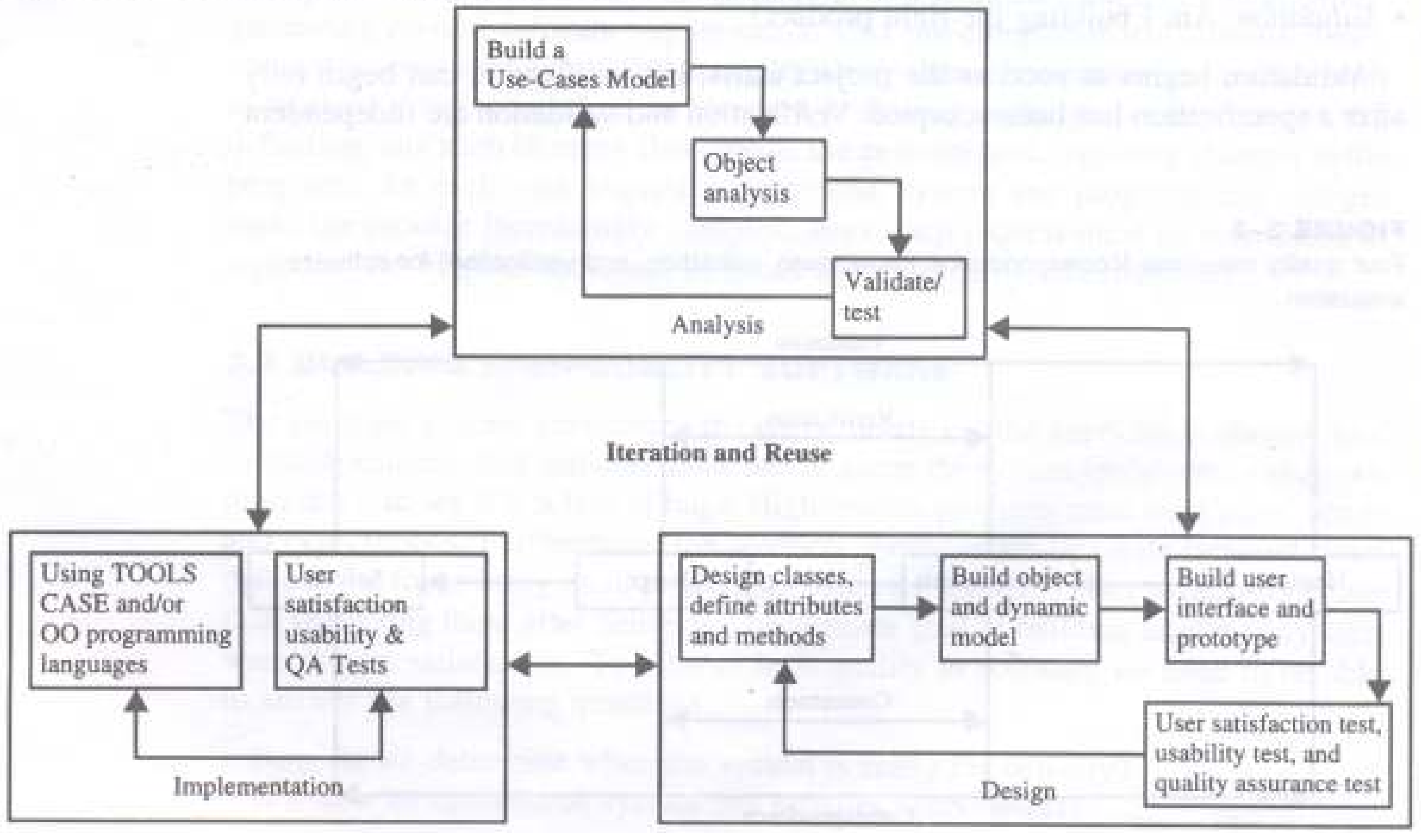
- Validation begins as soon as the project starts, but verification can begin only after a specification has been accepted.
- Verification and validation are independent of each other.
- It is possible to have a product that corresponds to the specification, but if the specification proves to be incorrect, we do not have the right product;

Object-Oriented Systems Development: A use case driven approach

- The object oriented software development life cycle(SDLC) consists of three macro processes:
  - Object-oriented analysis
  - Object-oriented design
  - Object-oriented implementation

Object-Oriented Systems Development: A use case driven approach

FIGURE 3–4
The object-oriented systems development approach. Object-oriented analysis corresponds to transformation 1; design to transformation 2, and implementation to transformation 3 of Figure 3–1.



Build a Use-Cases Model
Object analysis
Validate/ test
Analysis
Iteration and Reuse
Using TOOLS CASE and/or OO programming languages
User satisfaction usability & QA Tests
Implementation
Design classes, define attributes and methods
Build object and dynamic model
Build user interface and prototype
User satisfaction test, usability test, and quality assurance test
Design

Object-Oriented Systems Development: A use case driven approach

- The use case model can be employed throughout most activities of software development.
- Furthermore, by following the lifecycle model of Jacobson , Ericsson and Jacobson, one can produce designs that are traceable across requirements, analysis, design, implementation, and testing.

Object-Oriented Systems Development: A use case driven approach

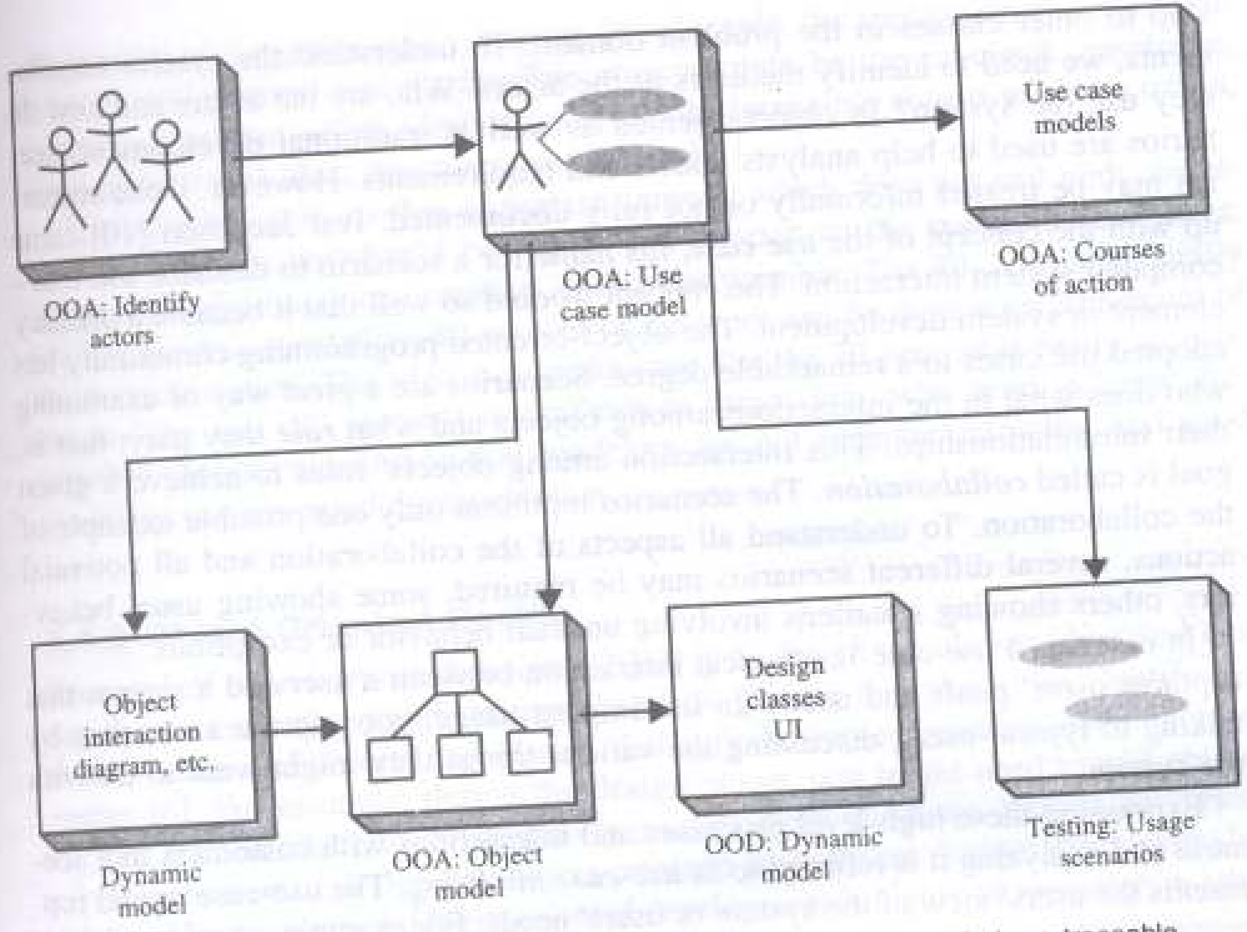


FIGURE 3-5
By following the life cycle model of Jacobson et al., we produce designs that are traceable across requirements, analysis, implementation, and testing.

Object-Oriented Systems Development: A use case driven approach

- The main advantage is that all design decisions can be tracked back directly to user requirements.
- Use case scenarios can become test scenarios.

Object-Oriented Systems Development: A use case driven approach

- Object oriented systems development includes these activities:
  - Object-oriented analysis- Use case driven
  - Object oriented design
  - Prototyping
  - Component-based development
  - Incremental testing

Object oriented Analysis- Use case Driven

- The object oriented analysis phase of software development is concerned with determining the system requirements and identifying classes and their relationship to other classes in the problem domain.
- To understand the system requirements, we need to identify the users or the actors.
- Who are the actors and how do they use the system?

Object oriented Analysis- Use case Driven

- Ivar Jacobson came up with the concept of use case, his name for a scenario to describe the user- computer system interaction.
- This concept worked so well that it became the primary element in the system development.
- The object-oriented community has adopted use cases to a remarkable degree.

Object oriented Analysis- Use case Driven

- Scenarios are a great way of examining who does what in the interactions among objects and what role they play; i.e., their interrelationships.
- This intersection among object's roles to achieve a given goal is called collaboration.
- A use case is a typical interaction between a user and system that captures user's goals and needs.

Object oriented Analysis- Use case Driven

- Expressing these high-level processes and interactions with customers in a scenario and analyzing it is referred to as use case modeling.
- The use case model represents the user's view of the system or user's needs.

Object oriented Analysis- Use case Driven

- This process of developing use cases, like other object-oriented activities, is iterative—once your use-case model is better understood and developed you should start to identify classes and create their relationships

Object oriented Analysis- Use case Driven

- Documentation is another important activity, which does not end with object-oriented analysis but should be carried out throughout the system development.
- Make the document as short as possible.
- The 80-20 rule generally applies for documentation: 80 % of the work can be done with 20% of documentation.
- Documentation and modeling are not separate activities, and good modeling implies good documentation.

Object-Oriented Design

- The goal of ***object-oriented design*** (OOD) is to design the classes identified during the analysis phase and the user interface.
- During this phase, we identify and define additional objects and classes that support implementation of the requirements.
- For example, during the design phase, you might need to add objects for the user interface to the system (e.g., data entry windows, browse windows).

Object-Oriented Design

- Object-oriented design and object-oriented analysis are distinct disciplines, but they can be intertwined.
- Object-oriented development is highly incremental; in other words, you start with object-oriented analysis, model it, create an object-oriented design, then do some more of each, again and again, gradually refining and completing models of the system:

Object-Oriented Design

- First, build the object model based on objects and their relationships, then iterate and refine the model:
  - Design and refine classes.
  - Design and refine attributes.
  - Design and refine methods.
  - Design and refine structures.
  - Design and refine associations.

Object-Oriented Design

- Here are a few guidelines to use in your object-oriented design:
  - Reuse, rather than build, a new class. Know the existing classes.
  - Design a large number of simple classes, rather than a small number of complex classes.
  - Design methods.
  - Critique what you have proposed. If possible, go back and refine the classes.

Prototyping

- Although the object-oriented analysis and design describe the system features, it is important to construct a prototype of some of the key system components shortly after the products are selected.
- It has been said "a picture may be worth a thousand words, but a prototype is worth a thousand pictures" [author unknown].

Prototyping

- Essentially, prototype is a version of a software product developed in the early stages of the product's life cycle for specific, experimental purposes.
- A prototype enables you understand how easy or difficult it will be to implement some of the features of the system.
- It also can give users a chance to comment on the usability and usefulness of the user interface design and lets you assess the fit between the software tools selected, the functional specification, and the user needs.

Prototyping

- Prototyping can further define the use cases, and it actually makes use-case modeling much easier.
- Traditionally, prototyping was used as a "quick and dirty" way to test the design, user interface, and so forth, something to be thrown away when the "industrial strength" version was developed.
- However, the new trend, such as using rapid application development, is to refine the prototype into the final product.

Prototyping

- Prototypes have been categorized in various ways:
  - A ***horizontal prototype*** is a simulation of the interface (that is, it has the entire user interface that will be in the full-featured system) but contains no functionality.
  - This has the advantages of being very quick to implement, providing a good overall feel of the system, and allowing users to evaluate the interface on the basis of their normal, expected perception of the system.

Prototyping

- A ***vertical prototype*** is a subset of the system features with complete functionality.
- The principal advantage of this method is that the few implemented functions can be tested in great depth.
- In practice, prototypes are a hybrid between horizontal and vertical.
- The major portions of the interface are established so the user can get the feel of the system, and features having a high degree of risk are prototyped with much more functionality

Prototyping

- An ***analysis prototype*** is an aid for exploring the problem domain.
- This class of prototype is used to inform, the user and demonstrate the proof of a concept.
- It is not used as the basis of development, however, and is discarded when it has served its purpose.
- The final product will use the concepts exposed by the prototype, not its code.

Prototyping

- A ***domain prototype*** is an aid for the incremental development of the ultimate software solution.
- It often is used as a tool for the staged delivery of subsystems to the users or other members of the development team.
- It demonstrates the feasibility of the implementation and eventually will evolve into a deliverable product.

Implement: Component Based Development

- Manufacturers learned that benefits of moving from custom development to assembly from prefabricated components.
- Components based manufacturing makes many products available to the market place that otherwise would be expensive.
- Modern manufacturing has evolved to exploit two crucial factors underlying today's market requirements: reduce cost and time to market by building from prebuilt, ready-tested components.

Component based development

- Today' software components are built and tested in-house, using a wide range of technologies like CASE TOOLS that allows
- ***Component-based development*** (CBD) is an industrialized approach to the software development process.
- Application development moves from custom development to assembly of prebuilt, pretested, reusable software components that operate with each other.

Component based development

- Two basic ideas underlie component-based development.
- First, the application development can be improved significantly if applications can be assembled quickly from prefabricated software components.
- Second, an increasingly large collection of interpretable software components could be made available to developers in both general and specialist catalogs.

Component based development

- A CBD developer can assemble components to construct a complete software system.
- Components themselves may be constructed from other components and so on down to the level of prebuilt components or old-fashioned code written in a language such as C, assembler, or COBOL.

Rapid application development

- ***Rapid application development*** (RAD) is a set of tools and techniques that can be used to build an application faster than typically possible with traditional methods.
- The term often is used in conjunction with software prototyping .
- It is widely held that, to achieve, RAD, the developer sacrifices the quality of the product for a quicker delivery.

Rapid application development

- This is not necessarily the case. RAD is concerned primarily with reducing the "time to market," not exclusively the software development time.
- In fact, one successful RAD application achieved a substantial reduction in time to market but realized no significant reduction in the individual software cycles

Rapid application development

- RAD does not replace the system development life cycle (see the Real-World case) but complements it, since it focuses more on process description and can be combined perfectly with the object-oriented approach.
- The task of RAD is to build the application quickly and incrementally implement the design and user requirements, through tools such as Delphi, VisualAge, Visual Basic, or PowerBuilder.

Incremental Testing

- That's what happened at Bankers Trust in 1992: "Our testing was very complete and good, but it was costing a lot of money and would add months onto a project," says Glenn Shimamoto, vice president of technology and strategic planning at the New York bank.

Incremental Testing

- In one case, testing added nearly six months to the development of a funds transfer application.
- The problem was that developers would turn over applications to a quality assurance (QA) group for testing only after development was completed.
- Since the QA group wasn't included in the initial plan, it had no clear picture of the system characteristics until it came time to test.

REUSABILITY

- A major benefit of object-oriented system development is reusability, and this is the most difficult promise to deliver on.
- For an object to be really reusable, much more effort must be spent designing it.
- To deliver a reusable object, the development team must have the up-front time to design reusability into the object.
- The potential benefits of reuse are clear: increased reliability, reduced time and cost for development, and improved consistency

REUSABILITY

- The reuse strategy can be based on the following:
  - Information hiding (encapsulation).
  - Conformance to naming standards.
  - Creation and administration of an object repository.
  - Encouragement by strategic management of reuse as opposed to constant redevelopment.
  - Establishing targets for a percentage of the objects in the project to be reused (i.e., 50 percent reuse of objects).