1.2 Different Approaches to Improving Development

Attempts to make systems development less of an art and more of a science are usually referred to as **systems engineering or software engineering**

- CASE TOOLS
- Rapid Application Development
- Service-Oriented Architecture
- Agile Methodologies
- eXtreme Programming
- Object-Oriented Analysis And Design

CASE TOOLS:

CASE tools have been developed for internal use and for sale by several leading firms. CASE tools are used to support a wide variety of SDLC activities. CASE tools can be used to help in multiple phases of the SDLC: project identification and selection, project initiation and planning, analysis, design, and implementation and maintenance. An integrated and standard database called a repository is the common method for providing product and tool integration, and has been a key factor in enabling CASE to more easily manage larger, more complex projects and to seamlessly (to be full of good material and ideas to use) integrate data across various tools and products. The idea of a central repository of information about a project is not new—the manual form of such a repository is called a project dictionary or workbook. The general types of CASE tools are listed below:

- Diagramming tools enable system process, data, and control structures to be represented graphically.
- Computer display and report generators help prototype how systems “look and feel.” Display (or form) and report generators make it easier for the systems analyst to identify

data requirements and relationships.

- Analysis tools automatically check for incomplete, inconsistent, or incorrect specifications in diagrams, forms, and reports.

A central repository enables the integrated storage of specifications, diagrams, reports, and project management information.

Documentation generators produce technical and user documentation in standard formats.

Code generators enable the automatic generation of program and database definition code directly from the design documents, diagrams, forms, and reports.

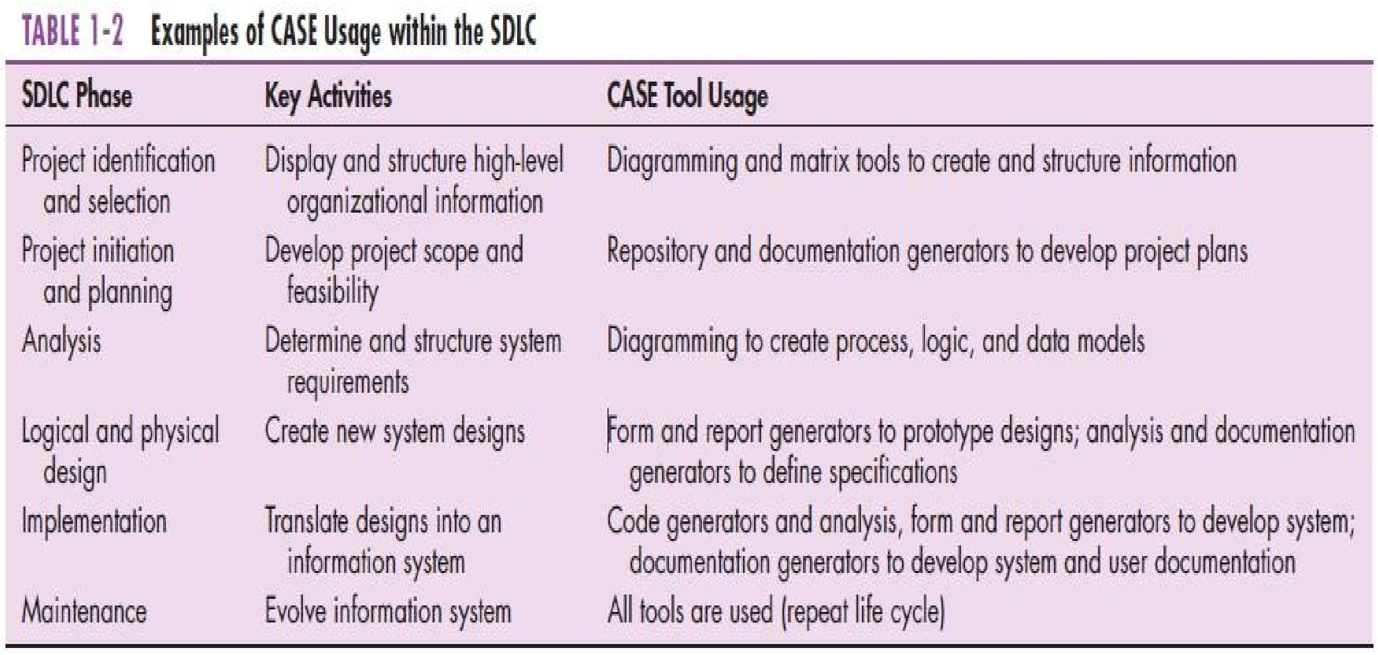


TABLE 1-2 Examples of CASE Usage within the SDLC

SDLC Phase	Key Activities	CASE Tool Usage
Project identification and selection	Display and structure high-level organizational information	Diagramming and matrix tools to create and structure information
Project initiation and planning	Develop project scope and feasibility	Repository and documentation generators to develop project plans
Analysis	Determine and structure system requirements	Diagramming to create process, logic, and data models
Logical and physical design	Create new system designs	Form and report generators to prototype designs; analysis and documentation generators to define specifications
Implementation	Translate designs into an information system	Code generators and analysis, form and report generators to develop system; documentation generators to develop system and user documentation
Maintenance	Evolve information system	All tools are used (repeat life cycle)

CASE helps programmers and analysts do their jobs more efficiently and more effectively by automating routine tasks. However, many organizations that use CASE tools do not use them to support all phases of the SDLC. Some organizations may extensively use the diagramming features but not the code generators.

Table 1-2 summarizes how CASE is commonly used within each SDLC phase. There are a variety of reasons why organizations choose to adopt CASE partially or not use it at all. These reasons range from a lack of vision for applying CASE to all aspects of the SDLC, to the belief that CASE technology will fail to meet an organization's unique system development needs. In some organizations, CASE has been extremely successful, whereas in others it has not.

CASE stands for **Computer Aided Software Engineering**. It means, development and maintenance of software projects with help of various automated software tools. CASE tools are set of software application programs, which are used to automate SDLC activities. CASE tools are used by software project managers, analysts and engineers to develop software system. There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools.

Use of CASE tools accelerates the development of project to produce desired result and helps to uncover flaws before moving ahead with next stage in software development.

Central Repository - CASE tools require a central repository, which can serve as a source of common, integrated and consistent information. Central repository is a central place of storage where product specifications, requirement documents, related reports and diagrams, other useful information regarding management is stored. Central repository also serves as data dictionary.

Upper Case Tools - Upper CASE tools are used in planning, analysis and design stages of SDLC.

Lower Case Tools - Lower CASE tools are used in implementation, testing and maintenance.

Integrated Case Tools - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation. CASE tools can be grouped together if they have similar functionality, process activities and capability of getting integrated with other tools.

Diagram tools - These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form. For example, Flow Chart Maker tool for creating state-of-the-art flowcharts.

Design Tools - These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using refinement techniques. These tools provides detailing of each module and interconnections among modules. For example, Animated Software Design.

Agile Methodologies:

According to Fowler (2003), the Agile Methodologies share three key principles:

(1) a focus on **adaptive** rather than **predictive** methodologies

(2) a focus on **people** rather than **roles**, and

(3) a focus on **self-adaptive** processes.

(1) Agile Methodologies share **iterative development** (Martin, 1999).

Iterative development focuses on the frequent productionof working versions of a system that have a subset of the total number of required features. Iterative development provides feedback to customers and developers alike.

(2) Systems **analyst or tester or manager**, are not as **important** as the **individuals** who fill those roles. Fowler argues that the application of engineering principles to systems development has resulted in aview of people as **interchangeable units** instead of a view of people as **talented** individuals, each bringing something unique to the development team.

(3) The Agile Methodologies promote a **self-adaptive** software development process. As software is developed, the process used to develop it should be refined and improved. Development teams can do this through a review process, often associated with the completion of iterations. The implication is that, as processes are adapted, one would not expect to find a single monolithic methodology within a given corporation or enterprise. Instead, one would find many variations of the methodology, each of which reflects the particular talents and experience of the team using it. Agile Methodologies are not for every project. Fowler (2003) recommends an agile or adaptive process if your project involves

# Unpredictable or dynamic requirements,

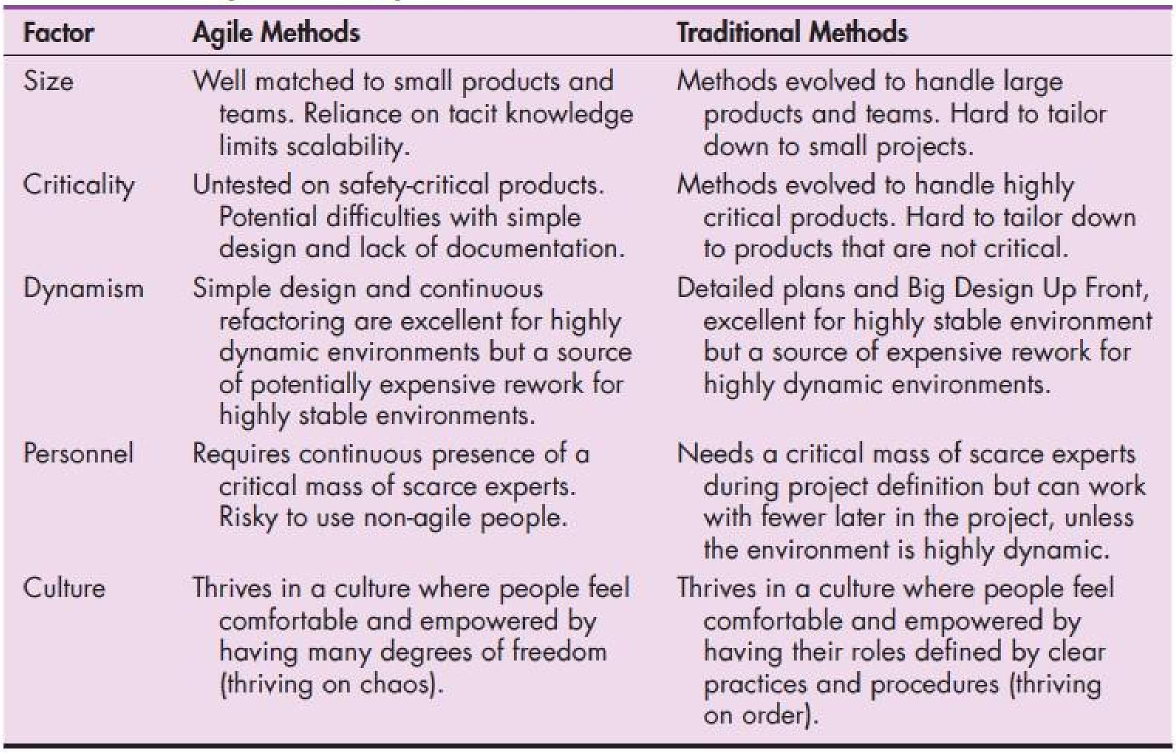
# Responsible and motivated developers, and

# Customers who understand the process and will get involved.

What is Agile?

- The word ‘agile’ means – Able to move your body quickly and easily.
- Able to think quickly and clearly.
- In business, ‘agile’ is used for describing ways of planning and doing work wherein it is understood that making changes as needed is an important part of the job. Business‘agililty’ means that a company is always in a position to take account of the market changes.
- In software development, the term ‘agile’ is adapted to mean ‘the ability to respond to changes – changes from Requirements, Technology and People.

TABLE 1-4 Five Critical Factors That Distinguish Agile and Traditional Approaches to Systems Development



Factor	Agile Methods	Traditional Methods
Size	Well matched to small products and teams. Reliance on tacit knowledge limits scalability.	Methods evolved to handle large products and teams. Hard to tailor down to small projects.
Criticality	Untested on safety-critical products. Potential difficulties with simple design and lack of documentation.	Methods evolved to handle highly critical products. Hard to tailor down to products that are not critical.
Dynamism	Simple design and continuous refactoring are excellent for highly dynamic environments but a source of potentially expensive rework for highly stable environments.	Detailed plans and Big Design Up Front, excellent for highly stable environment but a source of expensive rework for highly dynamic environments.
Personnel	Requires continuous presence of a critical mass of scarce experts. Risky to use non-agile people.	Needs a critical mass of scarce experts during project definition but can work with fewer later in the project, unless the environment is highly dynamic.
Culture	Thrives in a culture where people feel comfortable and empowered by having many degrees of freedom (thriving on chaos).	Thrives in a culture where people feel comfortable and empowered by having their roles defined by clear practices and procedures (thriving on order).

Prototyping

The Software Prototyping refers to building software application prototypes which display the functionality of the product under development but may not actually hold the exact logic of the original software. Software prototyping is becoming **very popular as a software development model,** as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

What is Software Prototyping?

Prototype is a **working model of software with some limited functionality.** The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation. Prototyping is used to allow the users **evaluate developer proposals and try them out before implementation.** It also helps understand the requirements which are user specific and may not have been considered by the developer during product design. Following is the stepwise approach to design a software prototype:

Basic Requirement Identification:

This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

Developing the initial Prototype:

The initial Prototype is developed in this stage, where the very basic requirements are show cased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed and the work arounds are used to give the same look and feel to the customer in the prototype developed.

Review of the Prototype:

The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

Revise and enhance the Prototype:

The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like, **time and budget constraints and technical feasibility of actual implementation.** The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until customer expectations are met. Prototypes can have **horizontal or vertical** dimensions. **Horizontal** prototype displays the **user interface for the product and gives a broader view** of the entire system, without concentrating on internal functions. A **vertical** prototype on the other side is a **detailed elaboration of a specific function or a sub system** in the product. The purpose of both horizontal and vertical prototype is different. Horizontal prototypes are used to get more information on the user interface level and the business requirements. It can even be presented in the sales demos to get business in the market. Vertical prototypes are technical in nature and are used to get details of the exact functioning of the sub systems. For example, database requirements, interaction and data processing loads in a given sub system. Software Prototyping Types There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely:

Throwaway/Rapid Prototyping:

Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

Evolutionary Prototyping:

Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. The prototype developed forms the heart of the future prototypes on top of which the entire system is built. Using evolutionary prototyping only well understood requirements are included in the prototype and the requirements are added as and when they are understood.

Incremental Prototyping:

Incremental prototyping refers to building multiple functional prototypes of the various sub systems and then integrating all the available prototypes to form a complete system.

Extreme Prototyping:

Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the html format. Then the data processing is simulated using a prototype services layer. Finally the services are implemented and integrated to the final prototype. This process is called Extreme Prototyping used to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the actual services.

Software Prototyping Application

Software Prototyping is most useful in development of systems having high level of user interactions such as online systems. Systems which need users to fill out forms or go through various screens before data is processed can use prototyping very effectively to give the exact look and feel even before the actual software is developed. Software that involves too much of data processing and most of the functionality is internal with very little user interface does not usually benefit from prototyping. Prototype development could be an extra overhead in such projects and may need lot of extra efforts.

Software Prototyping Pros and Cons

Software prototyping is used in typical cases and the decision should be taken very carefully so that the efforts spent in building the prototype add considerable value to the final software developed. The model has its own pros and cons discussed as below.

Following table lists out the pros and cons of Big Bang Model:

Pros	Cons
Increased user involvement in the product even before implementation , Since a working model of the system is displayed, the users get a better understanding of the system being developed.	**analysis owing to too much dependency on prototype.**
Reduces time and cost as the defects can be detected much earlier.	**Users may get confused in the prototypes and actual systems.**
Quicker user feedback is available leading to better solutions.	**Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.**
Risk of insufficient requirement	**Missing functionality can be identified easily Confusing or difficult functions can be**

identified.

Developers may try to reuse the existing prototypes to build the actual system, even when its not technically feasible

The effort invested in building prototypes may be too much if not monitored properly

eXtreme programming:

eXtreme Programming is an approach to software development put together by Beck and Andres (2004). It is distinguished by its short cycle **incremental planning approach focus on automated tests written by programmers and customers** to monitor the development process reliance on an **evolutionary approach** to development that lasts throughout the lifetime of the system use of two-person programming teams.

- **Planning**, **analysis**, **design**, and **construction** are all fused into a single phase of activity.
- Under this approach, **coding and testing are intimately related** parts of the same process. The programmers who write the code also develop the tests. The emphasis is on testing those things that can break or go wrong, not on testing everything.
- Code is tested very soon after it is written. The overall philosophy behind eXtreme Programming is that the code will be integrated into the system it is being developed for and tested within a few hours after it has been written. If all the tests run successfully,then

development proceeds. If not, the code is reworked until the tests are successful.

- Another part of eXtreme Programming that makes the code-and-test process work more smoothly is the practice of pair programming. All coding and testing is done by two people working together to write code and develop tests. **Beck** says that pair programming is not one person typing while the other one watches; rather, the two programmerswork together on the problem they are trying to solve, exchanging information and insight and sharing skills. Compared to traditional coding practices, the advantages of pair programming include:

(1)**more and better communication among developers**

(2)**higher levels of productivity**

(3)higher-quality code and

(4)**reinforcement of the other practices in eXtreme** programming, such as the code and- test discipline.

Although the eXtremeProgramming process has its advantages, just as with any other approach to systems development, it is not for everyone and is not applicable to every project.

- **Extreme Programming – A way to handle the common shortcomings**
- Software Engineering involves −
- Creativity
- Learning and improving through trials and errors

- Iterations
- Extreme Programming builds on these activities and coding. It is the detailed (not the only) design activity with multiple tight feedback loops through effective implementation, testing and refactoring continuously.
- Extreme Programming is based on the following values
- Communication
- Simplicity
- Feedback
- Courage
- Respect

What is Extreme Programming?

- XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.
- e**X**treme **P**rogramming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.
- Extreme Programming is one of the Agile software developmentmethodologies. It provides values and principles to guide the team behavior. The team is expected to self-organize. Extreme Programming provides specific core practices where –

- Each practice is simple and self-complete.
- Combination of practices produces more complex and emergent behavior.

Why is it called “Extreme?”

- Extreme Programming takes the effective principles and practices to extremelevels.
- Code reviews are effective as the code is reviewed all the time.
- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.

What is RAD?

- The **RAD (Rapid Application Development)** model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.
- Rapid Application Development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.
- Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product.
- In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.
- RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives

and other IT resources working progressively on their component or prototype.

- The most important aspect for this model to be successful is to make sure that the prototypes developed a reusable.

  Following are the main phases conducted within the RAD

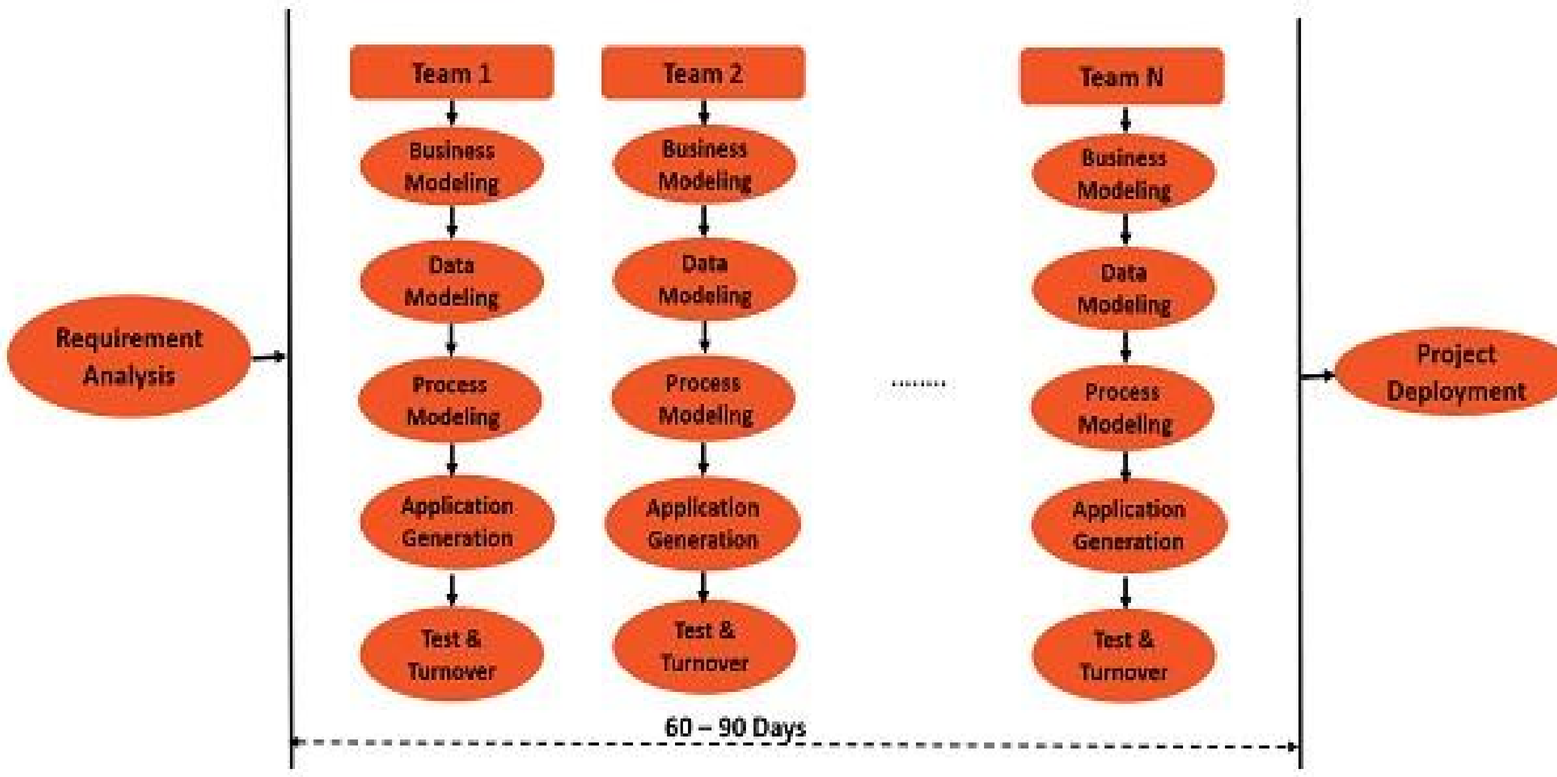
  Business Modelling

  Data Modelling

  Process Modelling

  Application Generation

  Testing and Turnover



Object-Oriented Analysis And Design:

- The object-oriented approach combines data and processes (**called methods**) into single entities called **objects. Objects usually** correspond to the real things an information system deals with, such as customers, suppliers, contracts, and rental agreements. Putting data and processes together in one place recognizes the fact that there are a limited number of operations for any given data structure, and the object-oriented approach makes sense even though typical systems development keeps data and processes independent of each other.

- The goal of OOAD is to make systems elements more reusable,by using **inheritance.** Inheritance allows the creation of new classes that share some of the characteristics of existing classes example, from a class of objects called "person," you can use inheritance to define another class of objects called "customer."thus improving system quality and the productivity of systems analysis and design.Objects of the class "customer" would share certain characteristics with objects of the class "person": They would both have names, addresses, phone numbers, and so on. Because "person" is the more general class and "customer" is more specific, every customer is a person but not every person is a customer.
- The object-oriented approach to systems development shares the **iterative development approach of the Agile Methodologies.**

- One of the most popular realizations of the iterative approach for object- oriented development is the **Rational Unified Process (RUP),** which is based on an **iterative**, **incremental** approach to systems development. RUP has four phases: **inception**, **elaboration**, **construction**, and **transition** (see Figure 1-11).
- In the **inception** phase, analysts define the scope, determine the feasibility of the project, understand user requirements, and prepare a software development plan.
- In the **elaboration** phase, analysts detail user requirements and develop a baseline architecture. Analysis and design activities constitute the bulk of the elaboration phase.
- In the **construction phase**, the software is actually coded, tested, and documented.
- In the **transition** phase, the system is deployed, and the users are trained and supported. As is evident from Figure 1-11, the construction phase is generally the longest and the most resource intensive. The elaboration phase is also long, but less resource intensive. The transition phase is resource intensive but short. The inception phase is short and the least resource intensive. The areas of the rectangles in Figure 1-11 provide an estimate of

the overall resource allocated to each phase.

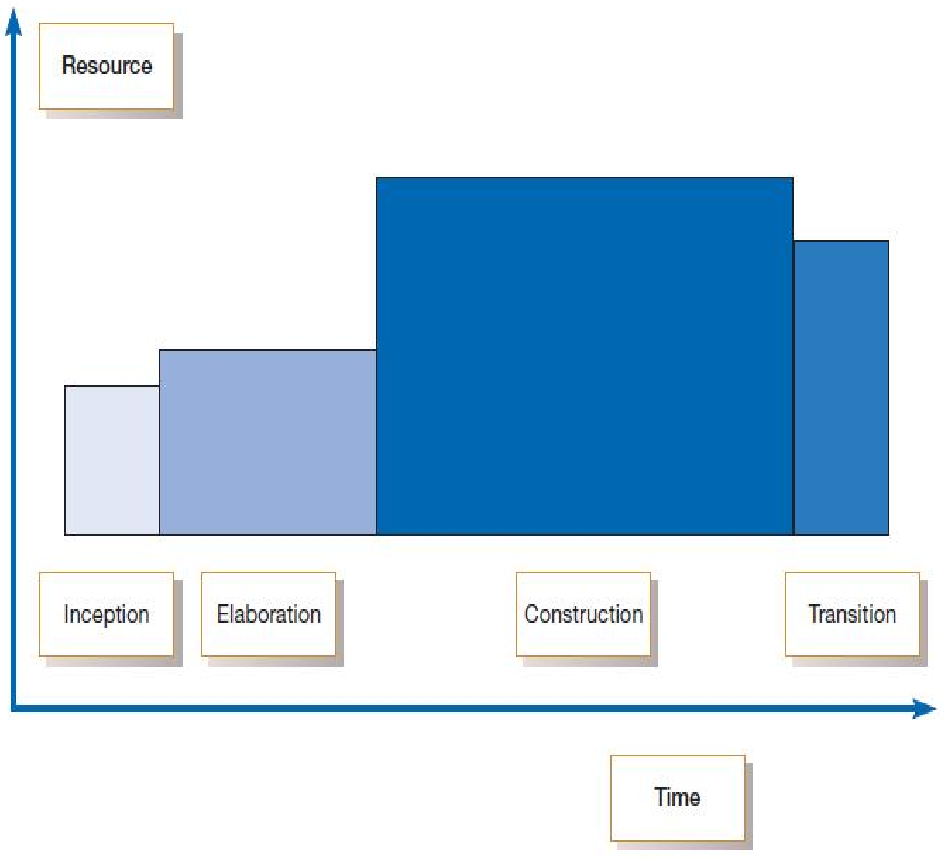


FIGURE 1-11
Phases of OOAD-based development

- Each phase can be further divided into iterations. The software is developed incrementally as a series of iterations. The **inception phase** will generally entail a **single** iteration. The scope and feasibility of the project is determined at this stage. The **elaboration** phase may have one or two iterations and is generally considered the most critical of the four phases (Kruchten, 2000). The elaboration phase is mainly about systems analysis and design, although other activities are also involved. At the end of the **elaboration phase, the architecture of the project should have been developed**. The architecture includes a vision of the product, an executable demonstration of the critical pieces, a detailed glossary and a preliminary user manual, a detailed construction plan, and a revised estimate of planned expenditures.