System Analysis and Design (SAD)

Chapter - 1

System

- System is an interrelated set of components, with identifiable boundary, working together for some purpose.
- Examples
- Solar system
- Digestive systems
- Public transport system
- Central heating system
- Computer system
- Information system

A set of objects and relationships among the objects viewed as a whole and designed to achieve a purpose

A system has nine characteristics:

1. Components or Subsystems
2. Interrelated components
3. A boundary
4. A purpose
5. An environment
6. Interfaces
7. Input
8. Output
9. Constraints

Component: An irreducible part or aggregation of parts that make up a system, also called a subsystem. A subsystem is simply a system within a system. Automobile is a system composed of subsystems:

- Engine system
- Body system
- Frame system
- Each of these subsystem is composed of sub-sub --systems.
- Engine system: carburetor system, generator system, fuel system, and so son

Interrelated components

- Dependence of one subsystem on one or more subsystems

Boundary

- The line that marks the inside and outside of a system and that sets off the system form its environment
- The overall goal or function of a system

Environment

- Everything external to a system that interacts with the system

Interface

- Point of contact where a system meets its environment or where subsystems meet each other.
- A limit to what a system can accomplish

Input

- Whatever a system takes from its environment in order to fulfill its purpose

Output

- Whatever a system returns from its environment in order to fulfill its purpose

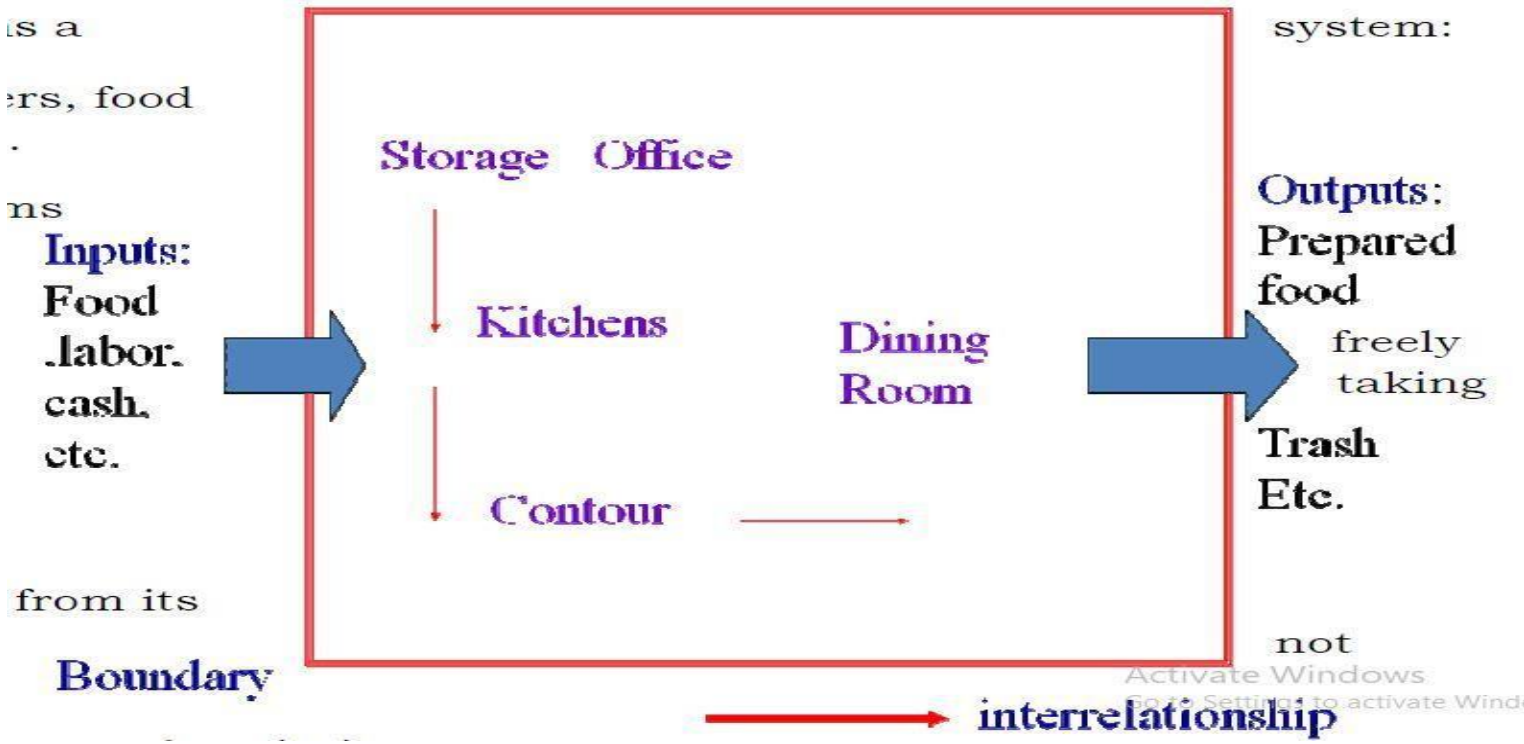
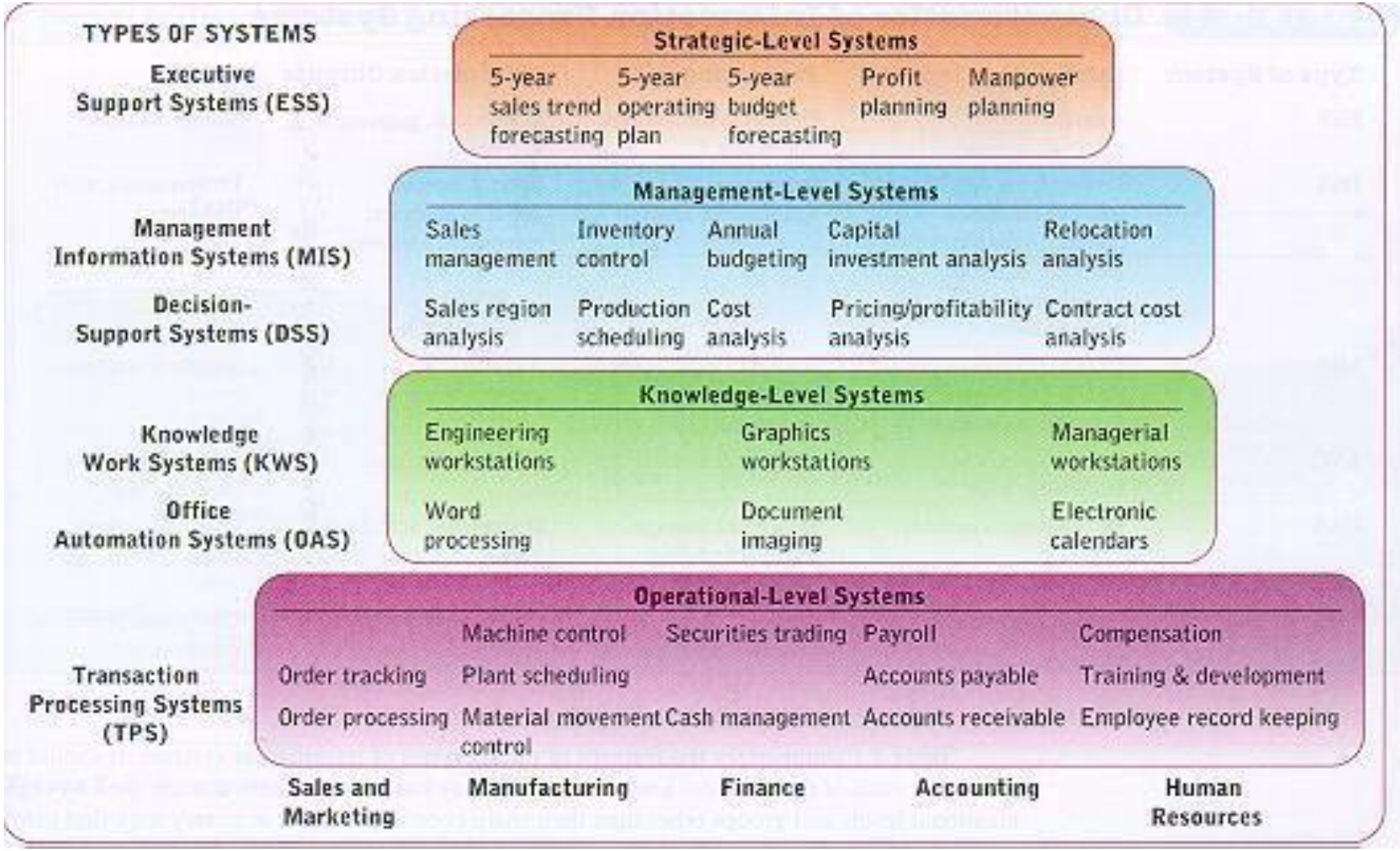


Fig: Fast food Restaurants as a system

What is an Information System?

- O'Brien Defines: Any organized combination of people, hardware, software, communications networks, and data resources that stores, retrieves, transforms, and disseminates information in an organization.
- HICKS Defins: IS as a formalized computer information system that can collect, store, process, and report data from various sources to provide the information necessary for management decision making.
- Laudon and Laudon (1995) defines: IS as a set of interrelated components that collect (or retrieve), process, store and disseminate information to support decision making, control, analysis and visualization in an organization.
- Components of IS includes
    - System, People Network, Software, Hardware and Data

Operational Level System



Transaction-Processing Systems (TPS)

- Basic business systems
- Perform daily routine transactions necessary for business functions
- At the operational level, tasks, resources and goals are predefined and highly structured
- Generally, five functional categories are identified, as shown in the diagram

Knowledge-level Systems

- **Office Automation Systems (OAS)**
- Targeted at meeting the knowledge needs of *data workers* within the organisation
- Data workers tend to process rather than create information. Primarily involved in information use, manipulation or dissemination.
- Typical OAS handle and manage documents, scheduling and communication.
- **Knowledge Work Systems (KWS)**
- Targeted at meeting the knowledge needs of *knowledge workers* within the organisation
- In general, knowledge workers hold degree-level professional qualifications (e.g. engineers, scientists, lawyers), their jobs consist primarily in creating new information and knowledge
- KWS, such as scientific or engineering design workstations, promote the creation of new knowledge, and its dissemination and integration throughout the organisation.

Management-level Systems

- **Management Information Systems (MIS)**
  - MIS provide managers with reports and, in some cases, on-line access to the organisations current performance and historical records
  - Typically these systems focus entirely on internal events, providing the information for short-term planning and decision making.
  - MIS summarise and report on the basic operations of the organisation, dependent on the underlying TPS for their data.
- **Decision-Support Systems (DSS)**
  - As MIS, these serve the needs of the management level of the organisation
  - Focus on helping managers make decisions that are semi-structured, unique, or rapidly changing, and not easily specified in advance
  - Use internal information from TPS and MIS, but also information from external sources
  - Greater analytical power than other systems, incorporate modelling tools, aggregation and analysis tools, and support *what-if* scenarios
  - Must provide user-friendly, interactive tools

Voyage-estimating Decision Support System

- **Strategic-level Systems**
- **Executive Support/Information Systems (ESS/EIS)**
- Serve the strategic level of the organisation
- ESS/EIS address unstructured decisions and create a generalised computing and communications environment, rather than providing any fixed application or specific capability. Such systems are not designed to solve specific problems, but to tackle a changing array of problems
- ESS/EIS are designed to incorporate data about external events, such as new tax laws or competitors, and also draw summarised information from internal MIS and DSS
- These systems filter, compress, and track critical data, emphasising the reduction of time and effort required to obtain information useful to executive management
- ESS/EIS employ advanced graphics software to provide highly visual and easy-to- use representations of complex information and current trends, but they tend not to provide analytical models

Modern Approach of system Analysis and Design

- **Information systems analysis and design** is a complex, challenging, and stimulating organizational process that a team of business and systems professionals uses to develop and maintain computer-based information systems. Although advances in information technology continually give us new capabilities, the analysis and design of information systems is driven from an organizational perspective.
- Information systems analysis and design is therefore an organizational improvement process. Systems are built and rebuilt for organizational benefits. Benefits result from adding value during the process of creating, producing, and supporting the organization's products and services. Thus the analysis and design of information systems is based on your understanding of the organization's objectives, structure, and processes, as well as your knowledge of how to exploit information technology for advantage.

- In the current business environment, the Internet, especially the World Wide Web, has been firmly integrated into an organization's way of doing business. Although you are probably most familiar with marketing done on the web and web-based retailing sites,
- such as eBay or Amazon.com, the overwhelming majority of business use of the web is business-to-business applications.
- Methodologies are comprehensive, multiple-step approaches to systems development that will guide your work and influence the quality of your final product— the information system. A methodology adopted by an organization will be consistent with its general management style (e.g., an organization's orientation toward consensus management will influence its choice of systems development methodology). Most methodologies incorporate several development techniques.

- Techniques are particular processes that you, as an analyst, will follow to help ensure that your work is well thought out, complete, and comprehensible to others on your project team. Techniques provide support for a
- wide range of tasks, including conducting thorough interviews to determine what your system should do, planning and managing the activities in a systems development project, diagramming the system's logic, and designing the reports your system will generate. Tools are typically computer programs that make it easy to use and benefit from techniques and to faithfully follow the guidelines of the overall development methodology.
- To be effective, techniques and tools must both be consistent with an organization's systems development methodology. Techniques and tools must make it easy for systems developers to conduct the steps called for in the methodology. These three elements—methodologies, techniques, and tools—work together to form an organizational approach to systems analysis and design (see Figure). Fig: An organizational approach to systems analysis and design is driven by methodologies, techniques, and tools.

DEVELOPING INFORMATION SYSTEMS AND THE SYSTEMS DEVELOPMENT LIFE CYCLE

- Most organizations find it beneficial to use a standard set of steps, called a **systems development methodology**, to develop and support their information systems. Like many processes, the development of information systems often follows a life cycle.
- For example, a commercial product follows a life cycle in that it is created, tested, and introduced to the market. Its sales increase, peak, and decline. Finally, the product is removed from the market and replaced by something else.
- The **systems development life cycle (SDLC)** is a common methodology for systems development in many organizations; it features several phases that mark the progress of the systems analysis and design effort.

System Development Life Cycle(SDLC)

1. System Planning

- The **systems planning phase** usually begins with a formal request to the IT department, called a **systems request**, which describes problems or desired changes in an information system or a business process. In many companies, IT systems planning is an integral part of overall business planning. When managers and users develop their business plans, they usually include IT requirements that generate systems requests. A systems request can come from a top manager, a planning team, a department head, or the IT department itself. The request can be very significant or relatively minor. A major request might involve a new information system or the upgrading of an existing system. In contrast, a minor request might ask for a new feature or a change to the user interface.

SYSTEMS PLANNING contd..

- The purpose of this phase is to perform a **preliminary investigation** to evaluate an IT-related business opportunity or problem. The preliminary investigation is a critical step because the outcome will affect the entire development process. A key part of the preliminary investigation is a **feasibility study** that reviews anticipated costs and benefits and recommends a course of action based on operational, technical, economic, and time factors.
- Suppose you are a systems analyst and you receive a request for a system change or improvement. Your first step is to determine whether it makes sense to launch a preliminary investigation at all. Often you will need to learn more about business operations before you can reach a conclusion. After an investigation, you might find that the information system functions properly, but users need more training. In some situations, you might recommend a business process review, rather than an IT solution. In other cases, you might conclude that a full-scale systems review is necessary. If the development process continues, the next step is the systems analysis phase.

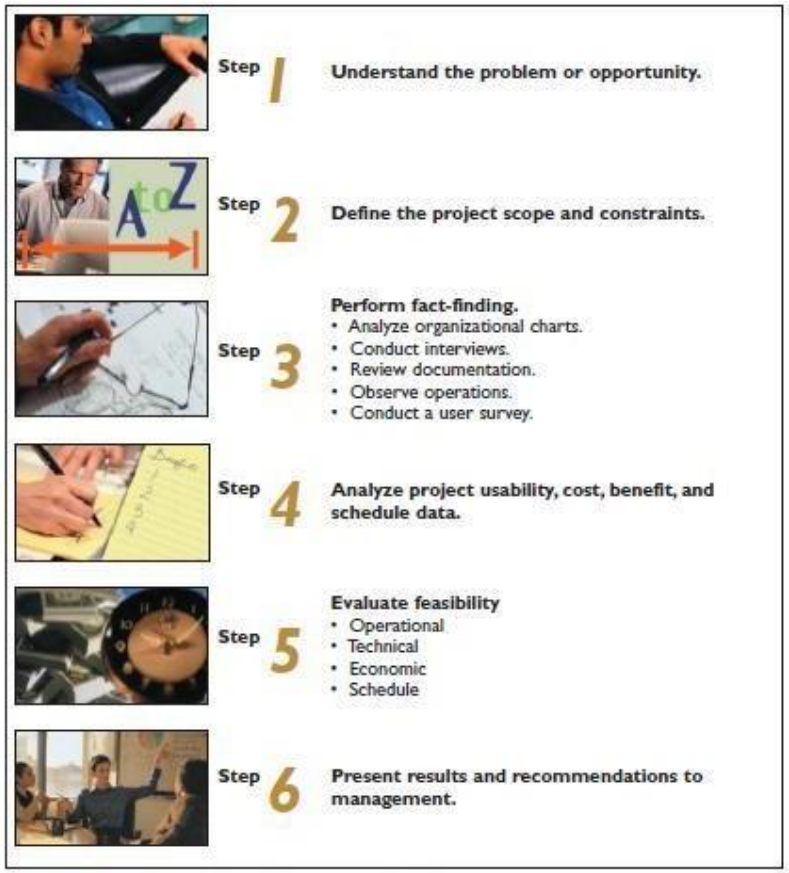


FIGURE 2-17 Six steps in a preliminary investigation.

2. SYSTEMS ANALYSIS

- The purpose of the **systems analysis phase** is to build a logical model of the new system. The first step is **requirements modeling**, where you investigate business processes and document what the new system must do to satisfy users.
- Requirements modeling continues the investigation that began during the systems planning phase. To understand the system, you perform fact-finding using techniques such as interviews, surveys, document review, observation, and sampling.
- You use the fact finding results to build business models, data and process models, and object models.
- The deliverable for the systems analysis phase is the **system requirements document**. The system requirements document describes management and user requirements, costs and benefits, and outlines alternative development strategies

System Analysis contd..

- **THE SYSTEMS ANALYST**
- A systems analyst investigates, analyzes, designs, develops, installs, evaluates, and maintains a company's information systems. To perform those tasks, a systems analyst constantly interacts with users and managers within and outside the company. On large projects, the analyst works as a member of an IT department team; on smaller assignments, he or she might work alone.
- Most companies assign systems analysts to the IT department, but analysts also can report to a specific user area such as marketing, sales, or accounting. As a member of a functional team, an analyst is better able to understand the needs of that group and how information systems support the department's mission. Smaller companies often use consultants to perform systems analysis work on an as-needed basis.

Responsibilities

- The systems analyst's job overlaps business and technical issues. Analysts help translate business requirements into IT projects. When assigned to a systems development team, an analyst might help document business profiles, review business processes, select hardware and software packages, design information systems, train users, and plan e-commerce Web sites.
- A systems analyst plans projects, develops schedules, and estimates costs. To keep managers and users informed, the analyst conducts meetings, delivers presentations, and writes memos, reports, and documentation.

Knowledge, Skills, and Education

- A successful systems analyst needs technical knowledge, oral and written communication skills, an understanding of business operations, and critical thinking skills. Educational requirements vary widely depending on the company and the position. In a rapidly changing IT marketplace, a systems analyst must manage his or her own career and have a plan for professional development.

3. SYSTEMS DESIGN

- The purpose of the **systems design phase** is to create a physical model that will satisfy all documented requirements for the system. At this stage, you design the user interface and identify necessary outputs, inputs, and processes. In addition, you design internal and external controls, including computer-based and manual features to guarantee that the system will be reliable, accurate, maintainable, and secure.
- During the systems design phase, you also determine the application architecture, which programmers will use to transform the logical design into program modules and code.
- The deliverable for this phase is the **system design specification**, which is presented to management and users for review and approval. Management and user involvement is critical to avoid any misunderstanding about what the new system will do, how it will do it, and what it will cost. critical to avoid any misunderstanding about what the new system will do, how it will do it, and what it will cost.

4. SYSTEMS IMPLEMENTATION

- During the **systems implementation phase,** the new system is constructed. Whether the developers use structured analysis or O-O methods, the procedure is the same — programs are written, tested, and documented, and the system is installed.
- If the system was purchased as a package, systems analysts configure the software and perform any necessary modifications. The objective of the systems implementation phase is to deliver a completely functioning and documented information system. At the conclusion of this phase, the system is ready for use.
- Final preparations include converting data to the new system's files, training users, and performing the actual transition to the new system. The systems implementation phase also includes an assessment, called a **systems evaluation**, to determine whether the system operates properly and if costs and benefits are within expectations.

SYSTEMS DEVELOPMENT GUIDELINES

- **Develop a Plan:** Prepare an overall project plan and stick to it. Complete the tasks in a logical sequence. Develop a clear set of ground rules and be sure that everyone on the team understands them clearly.
- **Involve Users and Listen Carefully to Them:** Ensure that users are involved in the development process, especially when identifying and modeling system requirements. When you interact with users, listen closely to what they are saying.
- **Use Project Management Tools and Techniques:** Try to keep the project on track and avoid surprises. Create a reasonable number of checkpoints — too many can be burdensome, but too few will not provide adequate control.
- **Develop Accurate Cost and Benefit Information:** Managers need to know the cost of developing and operating a system, and the value of the benefits it will provide. You must provide accurate, realistic cost and benefit estimates, and update them as necessary.
- **Remain Flexible:** Be flexible within the framework of your plan. Systems development is a dynamic process, and overlap often exists among tasks. The ability to react quickly is especially important when you are working on a system that must be developed rapidly.

What is methodology in SDLC?

- A methodology is a comprehensive plan to be followed, multiple-step approach to system development that will guide your work and influence the quality of information system.
- It Include the model that needs to be followed, plus the tools and techniques that need to be used. It can be purchased or home-grown. Methodology is purchased because many information system organisations cannot afford to dedicate staff to the development and continuous improvement of a home-grown methodology. Whilst, Methodology vendors have a vested interest in keeping their methodologies current with the latest business and technology trends.
- Methodology is written information in the form of books and other documents by detailing every activity, which needs to be implemented by system developers which includes documentation forms and reports that need to be provided by the project team.

Software Process

- **What is it?** When you work to build a product or system, it's important to go through a series of predictable steps—a road map that helps you create a timely, high-quality result. The road map that you follow is called a "software process."
- **Who does it?** Software engineers and their managers adapt the process to their needs and then follow it. In addition, the people who have requested the software have a role to play in the process of defining, building, and testing it.
- **Why is it important?** Because it provides stability, control, and organization to an activity that can, if left uncontrolled, become quite chaotic. However, a modern software engineering approach must be "agile." It must demand only those activities, controls, and work products that are appropriate for the project team and the product that is to be produced.
- **What are the steps?** At a detailed level, the process that you adopt depends on the software that you're building. One process might be appropriate for creating software for an aircraft avionics system, while an entirely different process would be indicated for the creation of a website.
- **What is the work product?** From the point of view of a software engineer, the work products are the programs, documents, and data that are produced as a consequence of the activities and tasks defined by the process.
- **How do I ensure that I've done it right?**

There are a number of software process assessment mechanisms that enable organizations to determine the "maturity" of their software process. However, the quality, timeliness, and long-term viability of the product you build are the best indicators of the efficacy of the process that you use. Software process presents a description of a process from some particular perspective as:

- Specification.
- Design.
- Validation.
- Evolution.

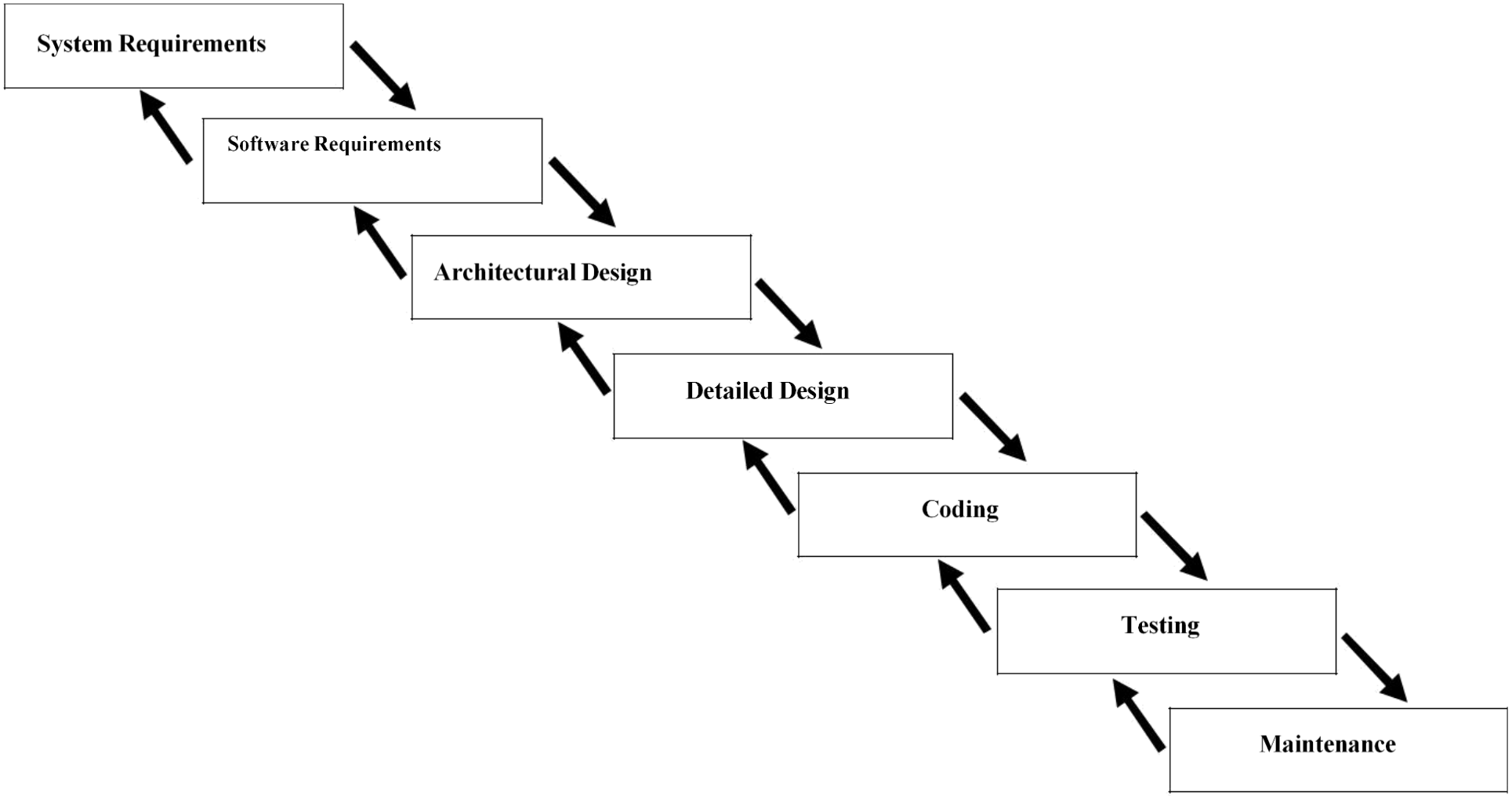
List of General Software Process Models

The list of traditional software development models are:

- Waterfall model
- Prototype model
- Rapid application development model
- Incremental model.
- Agile Model
- Iterative model.
- Spiral model

The Waterfall Model

- The waterfall model is the classical model of software engineering. This model is one of the oldest models and is widely used in government projects and in many major companies. As this model emphasizes planning in early stages, it ensures design flaws before they develop. In addition, its intensive document and planning make it work well for projects in which quality control is a major concern.
- The pure waterfall lifecycle consists of several non-overlapping stages, as shown in the following figure. The model begins with establishing system requirements and software requirements and continues with architectural design, detailed design, coding, testing, and maintenance.
- The waterfall model serves as a baseline for many other lifecycle models.


System Requirements
Software Requirements
Architectural Design
Detailed Design
Coding
Testing
Maintenance

- **System requirements**: Establishes the components for building the system, including the hardware requirements, software tools, and other necessary components. Examples include decisions on hardware, such as plug-in boards (number of channels, acquisition speed, and so on), and decisions on external pieces of software, such as databases or libraries.
- **Software requirements**: Establishes the expectations for software functionality and identifies which system requirements the software affects. Requirements analysis includes determining interaction needed with other applications and databases, performance requirements, user interface requirements, and so on.
- **Architectural design**: Determines the software framework of a system to meet the specific requirements. This design defines the major components and the interaction of those components, but it does not define the structure of each component. The external interfaces and tools used in the project can be determined by the designer.
- **Detailed design**: Examines the software components defined in the architectural design stage and produces a specification for how each component is implemented.
- "Architecture is concerned with the selection of architectural elements, their interaction, and the constraints on those elements and their interactions…Design is concerned with the modularization and detailed interfaces of the design elements, their algorithms and procedures, and the data types needed to support the architecture and to satisfy the requirements."
- **Coding**: Implements the detailed design specification.
- **Testing**: Determines whether the software meets the specified requirements and finds any errors present in the code.
- **Maintenance:** Addresses problems and enhancement requests after the software releases.

Advantages and disadvantages

Advantages:

- Easy to understand and implement.
- Widely used and known (in theory!).
- Reinforces good habits: define-before- design, design-before-code.
- Identifies deliverables and milestones.
- Document driven, URD, SRD, … etc. Published documentation standards
- Works well on mature products and weak teams.

Disadvantages:

- Idealized, doesn’t match reality well.
- Doesn’t reflect iterative nature of exploratory development.
- Unrealistic to expect accurate requirements so early in project.
- Software is delivered late in project, delays discovery of serious errors.
- Difficult to integrate risk management.
- Difficult and expensive to make changes to documents, ”swimming upstream”.
- Significant administrative overhead, costly for small teams and projects

The 5 Essential Stages of a RAD Model

- There are several stages to go through when developing a RAD model including analysis, designing, building, and the final testing phase. These steps can be divided to make them more easily understandable and achievable. The following describes the steps included in all RAD models:
- Stage 1: Business Modeling
- This step in the RAD model takes information gathered through many business related sources. The analysis takes all the pertinent information from the company. This info is then combined into a useful description of how the information can be used, when it is processed, and what is making this specific information successful for the industry.
- Stage 2: Data Modeling
- During the Data Modeling stage, all the information gathered during the Business Modeling phase is analyzed. Through the analysis, the information is grouped together into different groups that can be useful to the company. The quality of every group of information is carefully examined and given an accurate description. A relationship between these groups and their usefulness as defined in the Business Modeling step is also established during this phase of the RAD model.
- Stage 3: Process Modeling
- The Process Modeling phase is the step in the RAD model procedure where all the groups of information gathered during the Data Modeling step are converted into the required usable information. During the Process Modeling stage changes and optimizations can be done and the sets of data can be further defined. Any descriptions for adding, removing, or changing the data objects are also created during this phase.

Stage 4: Application Generation

- The Application Generation step is when all the information gathered is coded, and the system that is going to be used to create the prototype is built. The data models created are turned into actual prototypes that can be tested in the next step.

Stage 5: Testing and Turnover

- The Testing and Turnover stage allows for a reduced time in the overall testing of the prototypes created. Every model is tested individually so that components can quickly be identified and switched in order to create the most effective product. By this point in the RAD model, most of the components have already been examined, so major problems with the prototype are not likely.

Iterative Development:

- The problems with the Waterfall Model created a demand or a new method of developing
- systems which could provide faster results, require less up-front information, and offer greater flexibility. With Iterative Development, the project is divided into small parts.
- This allows the development team to demonstrate results earlier on in the process and obtain valuable feedback from system users.
- Often, each iteration is actually a mini-Waterfall process with the feedback from one phase providing vital information for the design of the next phase. In a variation of this model, the
- software products, which are produced at the end of each step (or series of steps), can go into production immediately as incremental releases

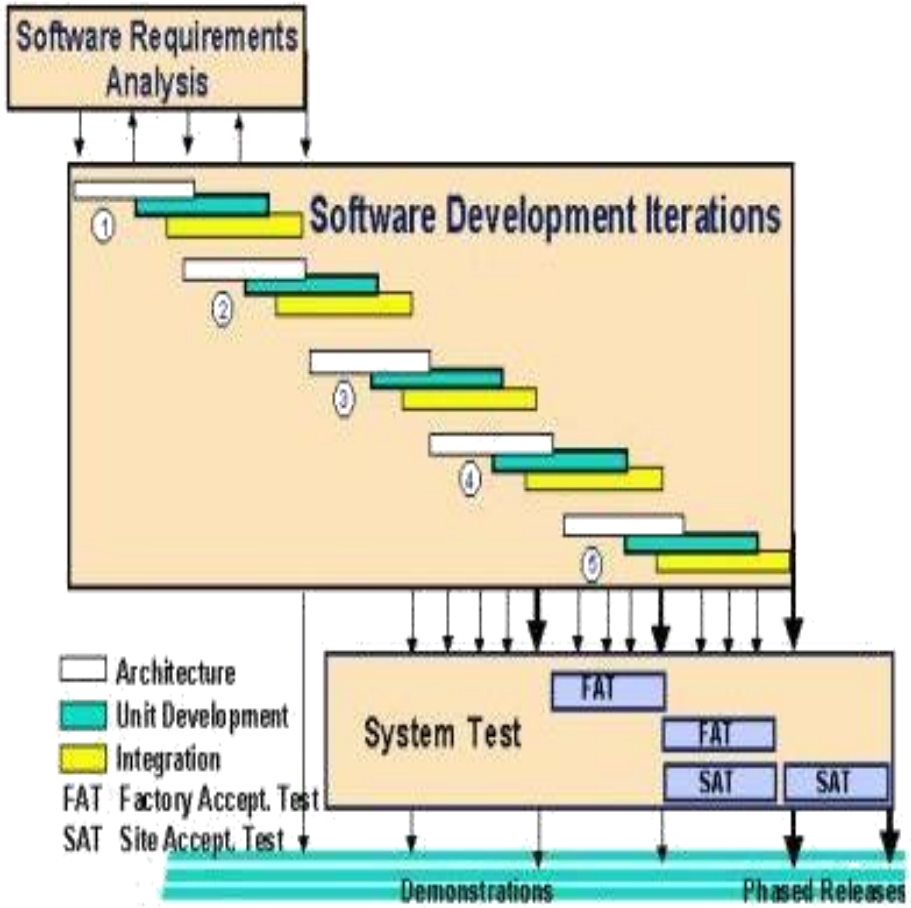
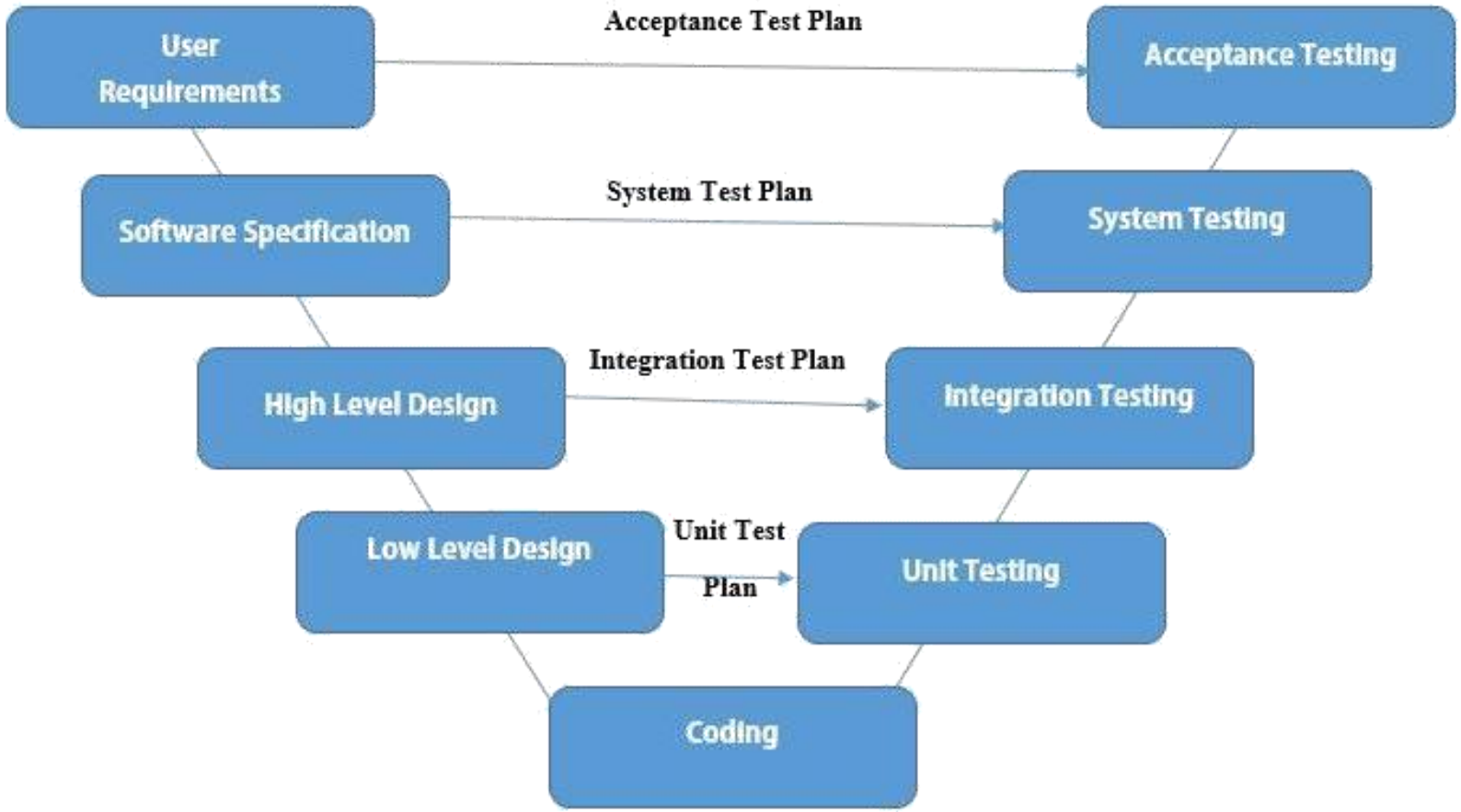


Fig. 4 Iterative Development.

V-Shaped Model

- Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing is emphasized in this model more than the waterfall model.
- The testing procedures are developed early in the life cycle before any coding is done, during each of the phases preceding implementation. Requirements begin the life cycle model just like the waterfall model. Before development is started, a system test plan is created.
- The test plan focuses on meeting the functionality specified in requirements gathering. The high-level design phase focuses on system architecture and design.
- An integration test plan is created in this phase in order to test the pieces of the software systems ability to work together. However, the low-level design phase lies where the actual software components are designed, and unit tests are created in this phase as well.
- The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.


User Requirements
Acceptance Test Plan
Acceptance Testing
Software Specification
System Test Plan
System Testing
High Level Design
Integration Test Plan
Integration Testing
Low Level Design
Unit Test Plan
Unit Testing
Coding

V-Model

Advantages and Disadvantages

Advantages

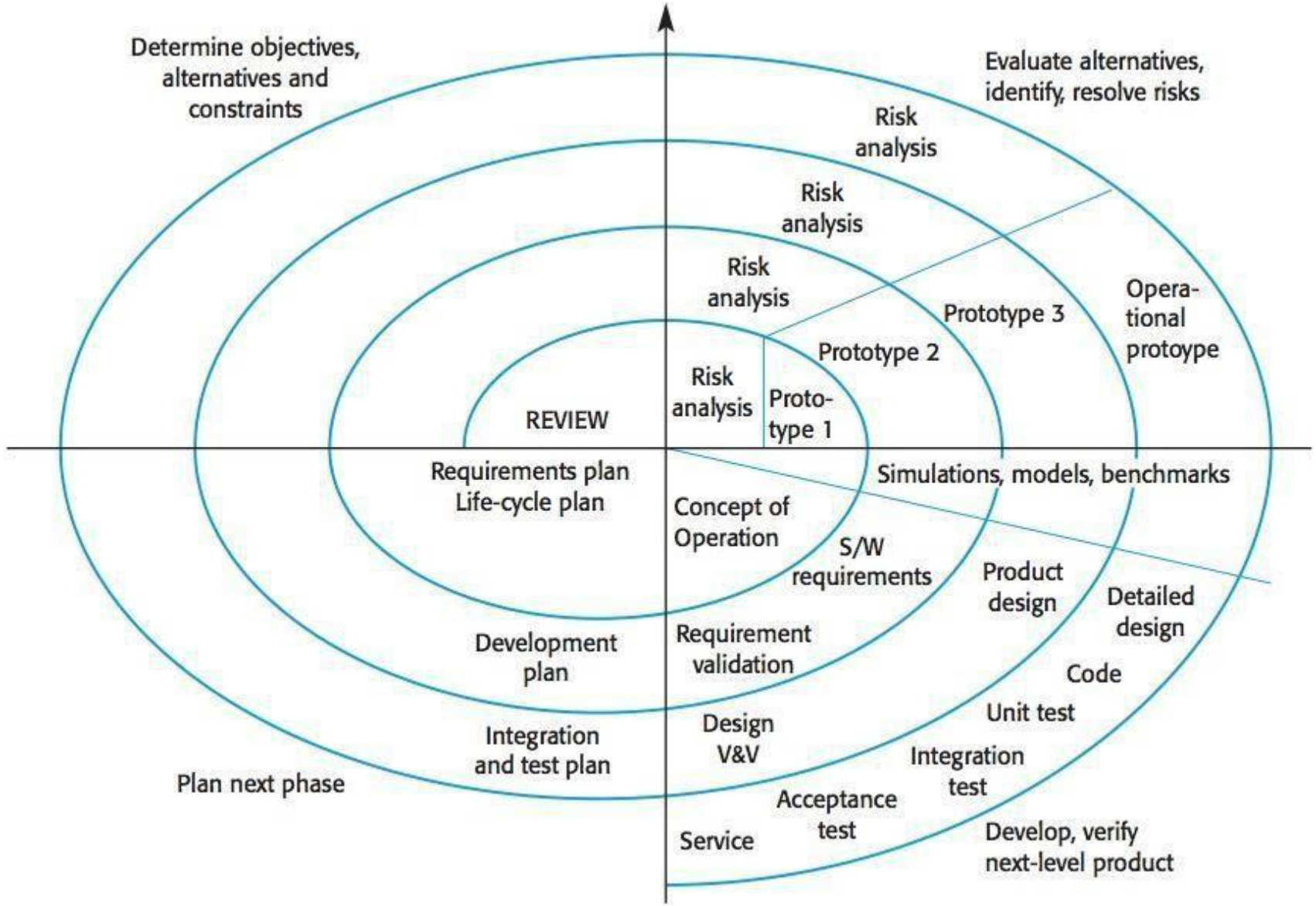
- Simple and easy to use.
- Each phase has specific deliverables.
- Higher chance of success over the waterfall model due to the early development of test plans during the life cycle.
- Works well for small projects where requirements are easily understood.

Disadvantages

- Very rigid like the waterfall model.
- Little flexibility and adjusting scope is difficult and expensive.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- This Model does not provide a clear path for problems found during testing phases.

Spiral Model

- The spiral model is similar to the incremental model, with more emphases placed on risk analysis. The spiral model has four phases: **Planning, Risk Analysis, Engineering and Evaluation**.
- A software project repeatedly passes through these phases in iterations (called Spirals in this model).
- The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral. Requirements are gathered during the planning phase. In the risk analysis phase, a process is undertaken to identify risk and alternate solutions.
- A prototype is produced at the end of the risk analysis phase. Software is produced in the engineering phase, along with testing at the end of the phase. The evaluation phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.
- In the spiral model, the angular component represents progress, and the radius of the spiral represents cost.


Determine objectives, alternatives and constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analysis
Prototype 3
Prototype 2
Operational protoype
REVIEW
Proto-type 1
Requirements plan
Life-cycle plan
Simulations, models, benchmarks
Concept of Operation
S/W requirements
Product design
Detailed design
Development plan
Requirement validation
Code
Unit test
Integration and test plan
Design V&V
Integration test
Plan next phase
Acceptance test
Service
Develop, verify next-level product

Advantages and Disadvantages

Advantages

- High amount of risk analysis.
- Good for large and mission-critical projects.
- Software is produced early in the software life cycle.

Disadvantages

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects

In Summary, spiral consists of:

- Planning: Define objectives, constraints, and deliverables
- Risk: analysis Identify risks and develop acceptable resolutions Engineering: Develop a prototype that includes all deliverables
- Evaluation: Perform assessment and testing to develop objectives for next iteration

Extreme Programming:

- An approach to development, based on the development and delivery of very small increments of functionality.
- It relies on constant code improvement, user involvement in the development team and pair wise programming. It can be difficult to keep the interest of customers who are involved in the process. Team members may be unsuited to the intense involvement that characterizes agile methods.
- Prioritizing changes can be difficult where there are multiple stakeholders. Maintaining simplicity requires extra work. Contracts may be a problem as with other approaches to iterative development.
- **Extreme Programming Practices Incremental planning**: Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development "Tasks".
- **Small Releases**: The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.

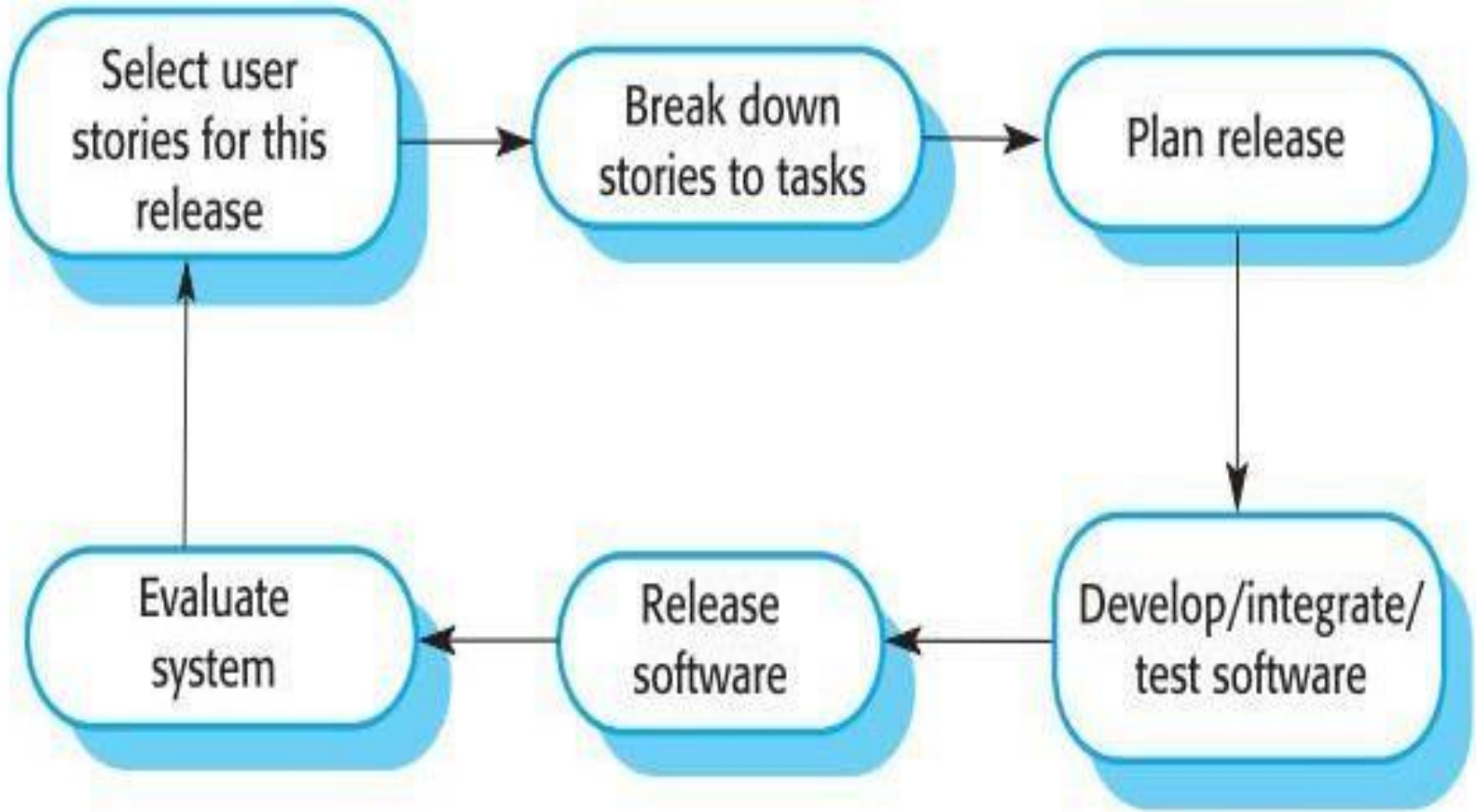


Fig. 8 The XP Release Cycle

- **Simple Design**: Enough design is carried out to meet the current requirements and no more.
- **Test first development**: An automated unit test framework is used to write tests for a new piece of functionality before functionality itself is implemented. Refactoring: All developers are expected to re-factor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.
- **Pair Programming**: Developers work in pairs, checking each other's work and providing support to do a good job.
- **Collective Ownership**: The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.
- **Continuous Integration**: As soon as work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
- **Sustainable pace:** Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity.
- **On-site Customer**: A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code

Advantages and Disadvantages

Advantages

- Lightweight methods suit small-medium size projects.
- Produces good team cohesion.
- Emphasizes final product.
- Iterative.
- Test based approach to requirements and quality assurance.

Disadvantages

- Difficult to scale up to large projects where documentation is essential.
- Needs experience and skill if not to degenerate into code-and-fix.
- Programming pairs is costly.
- Test case construction is a difficult and specialized skill.

DIFFERENT APPROACHES TO IMPROVING DEVELOPMENT

- In the continuing effort to improve the systems analysis and design process, several different approaches have been developed. Attempts to make systems development less of an art and more of a science are usually referred to as *systems engineering* or *software engineering*. As the names indicate, rigorous engineering techniques have been applied to systems development. One manifestation of the engineering approach is CASE tools.
- Other efforts to improve the systems development process have taken advantage of the benefits offered by computing technology itself.

CASE Tools

- The result has been the creation and fairly widespread use of **computer-aided software engineering (CASE) tools**. CASE tools have been developed for internal use and for sale by several leading firms, but the best known is the series of Rational tools made by IBM.
- 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 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.

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)

Table 1: EXAMPLES OF CASE USAGE WITHIN THE SDLC

AGILE METHODOLOGIES

- Many approaches to systems analysis and design have been developed over the years. In February 2001, many of the proponents of these alternative approaches met in Utah and reached a consensus on several of the underlying principles their various approaches contained. This consensus turned into a document they called “The Agile Manifesto”.
- According to Fowler (2003), the Agile Methodologies share three key principles:
  - a focus on adaptive rather than predictive methodologies,
  - a focus on people rather than roles, and
  - a focus on self-adaptive processes.

Contd…

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.

The Manifesto for Agile Software Development, *Seventeen anarchists agree:* We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan. That is, while we value the items on the right, we value the items on the left more.

We follow the following principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Businesspeople and developers work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design enhances agility.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

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).

- Many different individual methodologies come under the umbrella of Agile Methodologies.
- Fowler (2003) lists the Crystal family of methodologies, Adaptive Software Development, Scrum, Feature Driven Development, and others as Agile Methodologies.
- Perhaps the best known of these methodologies, however, is eXtreme Programming

Capability Maturity Model (CMM)

- The Capability Maturity Model (CMM) is a methodology used to develop and refine an organization's software development process. The model describes a five-level evolutionary path of increasingly organized and systematically more mature processes.
- CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center sponsored by the U.S. Department of Defense (DoD).
- SEI was founded in 1984 to address software engineering issues and, in a broad sense, to advance software engineering methodologies.
- More specifically, SEI was established to optimize the process of developing, acquiring, and maintaining heavily software-reliant systems for the DoD. Because the processes involved are equally applicable to the software industry as a whole, SEI advocates industry-wide adoption of the CMM.

CMM Level

- At the *initial* level, processes are disorganized, even chaotic. Success is likely to
- depend on individual efforts, and is not considered to be repeatable, because processes would not be sufficiently defined and documented to allow them to be replicated.
- At the *repeatable* level, basic project management techniques are established, and successes could be repeated, because the requisite processes would have been made established, defined, and documented.
- At the *defined* level, an organization has developed its own standard software process through greater attention to documentation, standardization, and integration.
- At the *managed* level, an organization monitors and controls its own processes through data collection and analysis.
- At the *optimizing* level, processes are constantly being improved through monitoring feedback from current processes and introducing innovative processes to better serve the organization's particular needs.

Alternate development methodologies

- **Development Strategies**

Analysts and organizations are increasingly faced with a make, buy or outsource decision when assessing development strategies for information systems project.

As an analyst, part of the expertise you are developing is to make sound judgments regarding developing software versus purchase of software or outsourcing for new and existing system.

Each alternative comes with its own strengths and weaknesses. Therefore, one alternative is better than others in different situations.

- **Create Custom Software**

Custom Software is also known as in-house software, is a type of software that is developed from the beginning for a specific organization or function. Most of the project groups consider custom development as the best approach to develop a system because the team has complete control over the functions and shape of the system.

- There are several situations that call for the custom development of a new software or software components. The most likely instance is when commercial off-the-shelf (COTS) software does not exist or cannot be identified for desired application. Alternatively, the software may exist but is unaffordable or cannot easily be purchased or licensed.
- Custom software development should be done when the organization is attempting to gain a competitive advantage through the leveraged use of information systems.
- This is often the case when an organization is creating ecommerce or other innovative applications where none existed. It is also possible that the organization is a “first pioneer” in the use of a particular technology or in its particular industry.
- Organizations that have highly specialized requirements or exist in niche industries can also benefit from custom development.
- Given Table summarizes the advantages and disadvantages of custom development.

Advantages	Disadvantages
• Specific response to specialised business needs. • Innovation may give firm a competitive advantage. • In-house staff available to maintain software. • Pride of ownership.	• May be significantly higher initial cost compared to COTS software or Application Service Provider. • Necessity of hiring or working with a development team. • Ongoing maintenance.

Purchasing COTS Packages

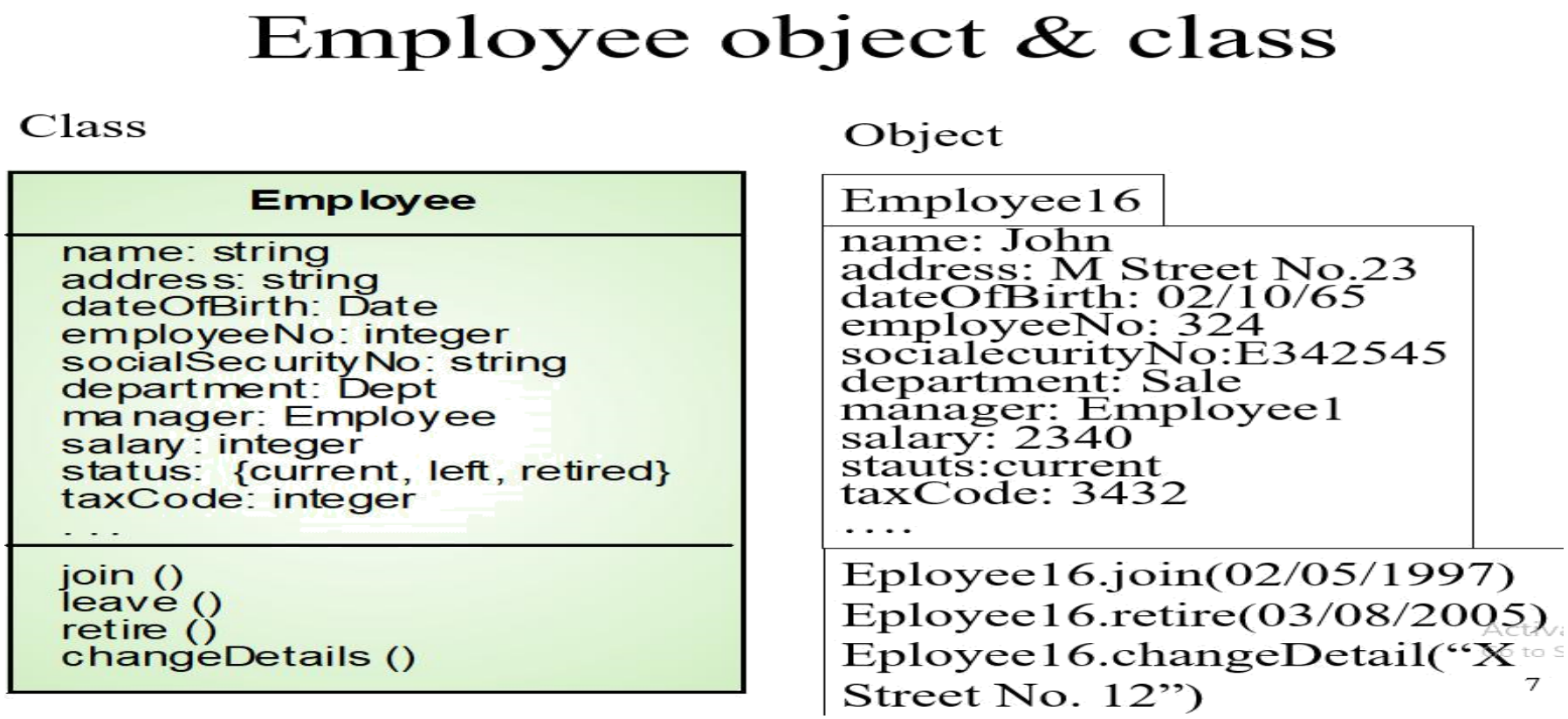
- Commercial Off-The-Shelf (COTS) software is a ready-made software product that can be easily obtained. COTS include such products as Microsoft Office suite which includes Word for word processing, Excel for spreadsheets, Access for building databases and other applications. Other types of COTS software from Enterprise Resource Planning (ERP) packages such as Oracle and SAP.
- Consider using COTS software when you can easily integrate the applications or packages into existing or planned systems, and when you have identified no necessity to immediately or continuously change or customise them for users.
- There are some advantages to purchasing COTS software that you should keep in mind as you weigh alternative.
- One advantage is that these products have been refined through the process of commercial use and distribution, so that often there are additional functionalities offered.
- Another advantage is that packaged software is typically extensively tested, and thus extremely reliable. Table given below summarizes the advantages and disadvantages of purchasing COTS packages
- Most of the COTS packages allow customization or manipulation of system parameters for changing certain functions. These changes are good in creating functions that are needed, but the said functions might not be present inside the software package.

Advantages	Disadvantages
• Refined in the commercial world. • Increased reliability. • Increased functionality. • Often lower initial cost. • Already in use by other firms. • Help and training comes with software.	• Programming focused; not business focused. • Must live with the existing features. • Limited customisation. • Uncertain financial future of vendor. • Less ownership and commitment.

Object Oriented Analysis and Design

- Object
  - Object is an abstraction of something in a problem domain, reflecting the capabilities of the system to keep information about it, interact with it, or both.
  - Objects are entities in a software system which represent instances of real-world and system entities
- Term of objects
  - Attribute: data items that define object.
  - Operation: function in a class that combine to form behavior of class.
  - Methods: the actual implementation of procedure (the body of code that is executed in response to a request from other objects in the system).

Difference between class and object



Identifying Object

- Objects can be:
  - External Entity (e.g., other systems, devices, people) that produce or consume information to be used by system
  - Things (e.g., reports, displays, letters, signals) that are part of information domain for the problem
  - Places (e.g., book's room) that establish the context of the problem and the overall function of the system.
  - Organizational units (e.g., division, group, team, department) that are relevant to an application,
  - Transaction (e.g., loan, take course, buy, order).

OO Analysis

- examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain. In other words, the world (of the system) is modelled in terms of objects and classes.
- In the object-oriented analysis, we perform the following activities:
- Elicit *requirements:* Define what does the software need to do, and what's the problem the software trying to solve.
- Specify *requirements*: Describe the requirements, usually, using use cases (and scenarios) or user stories.
- Conceptual *model*: Identify the important objects, refine them, and define their relationships and behavior and draw them in a simple diagram.

OO Design

- OO decomposition and a notation for depicting models of the system under development. Structures are developed whereby sets of objects collaborate to provide the behaviors that satisfy the requirements of the problem.
- In the object-oriented design, we …
- **Describe the classes** and their relationships using class diagram.
- **Describe the interaction** between the objects using sequence diagram.
- **Apply** software design principles and design patterns.

Software Engineering Development Activities

- Development activities deal with the complexity by constructing and validating models of the application domain or the system. An overview of the technical activities associated with object- oriented software engineering are:-
- Requirements Elicitation
- Analysis
- System Design
- Object Design
- Implementation
- Testing

Requirements Elicitation

- During **requirements elicitation**, the client and developers define the purpose of the system. The result of this activity is a description of the system in terms of actors and use cases.
- Actors represent the external entities that interact with the system.
- Actors include roles such as end users, other computers the system needs to deal with (e.g., a central bank computer, a network), and the environment (e.g., a chemical process).
- Use cases are general sequences of events that describe all the possible actions between an actor and the system for a given piece of functionality. Figure 1-3 depicts a use case for the TicketDistributor example.

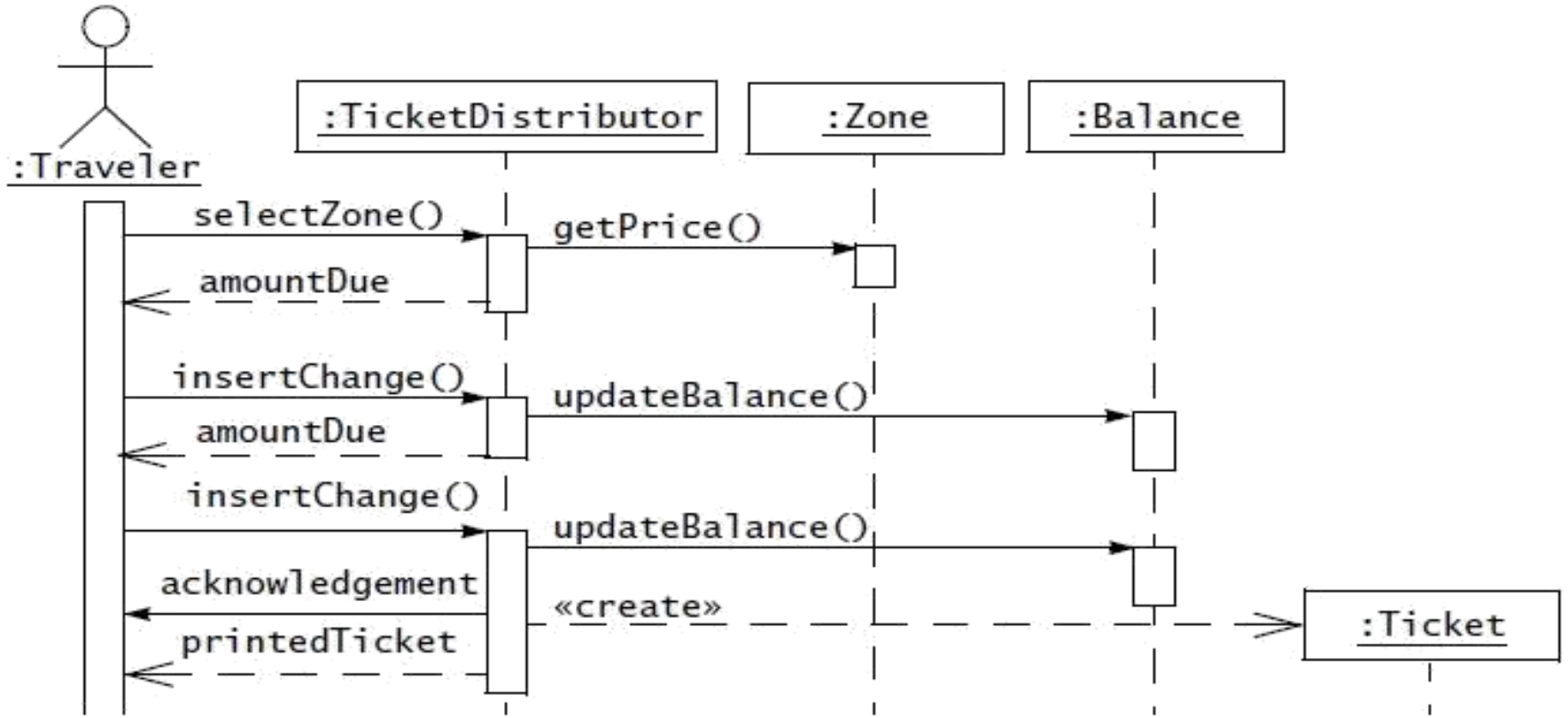


Figure 1-4 A dynamic model for the `TicketDistributor` (UML sequence diagram). This diagram depicts the interactions between the actor and the system during the `PurchaseOneWayTicket` use case and the objects that participate in the use case.

system design

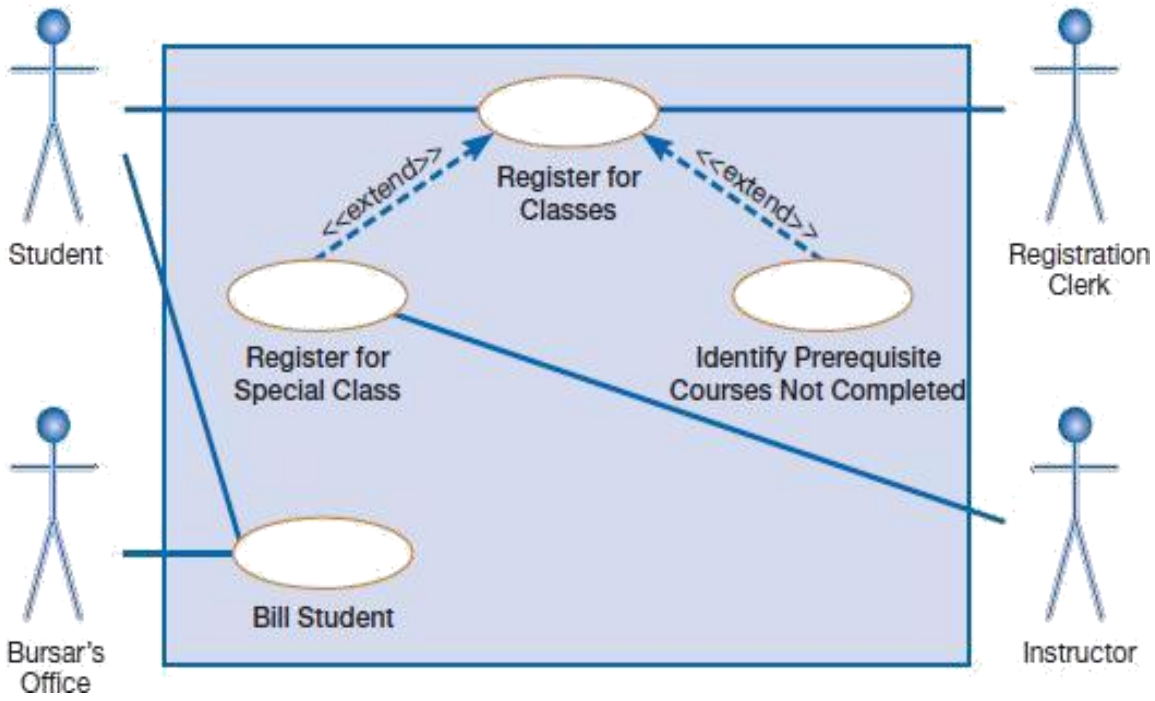
- During **system design**, developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams.
- Developers also select strategies for building the system, such as the hardware/software platform on which the system will run, the persistent data management strategy, the global control flow, the access control policy, and the handling of boundary conditions.
- The result of system design is a clear description of each of these strategies, a subsystem decomposition, and a deployment diagram representing the hardware/software mapping of the system.
- Whereas both analysis and system design produce models of the system under construction, only analysis deals with entities that the client can understand.
- System design deals with a much more refined model that includes many entities that are beyond the comprehension (and interest) of the client. Figure above depicts an example of system decomposition for the TicketDistributor.

Object Design and Implementation

- During **object design**, developers define solution domain objects to bridge the gap between the analysis model and the hardware/software platform defined during system design.
- This includes precisely describing object and subsystem interfaces, selecting off-the-shelf components, restructuring the object model to attain design goals such as extensibility or understandability, and optimizing the object model for performance.
- The result of the object design activity is a detailed object model annotated with constraints and precise descriptions for each element.
- During **implementation**, developers translate the solution domain model into source code.
- This includes implementing the attributes and methods of each object and integrating all the objects such that they function as a single system.
- The implementation activity spans the gap between the detailed object design model and a complete set of source code files that can be compiled.

What is a Use case?

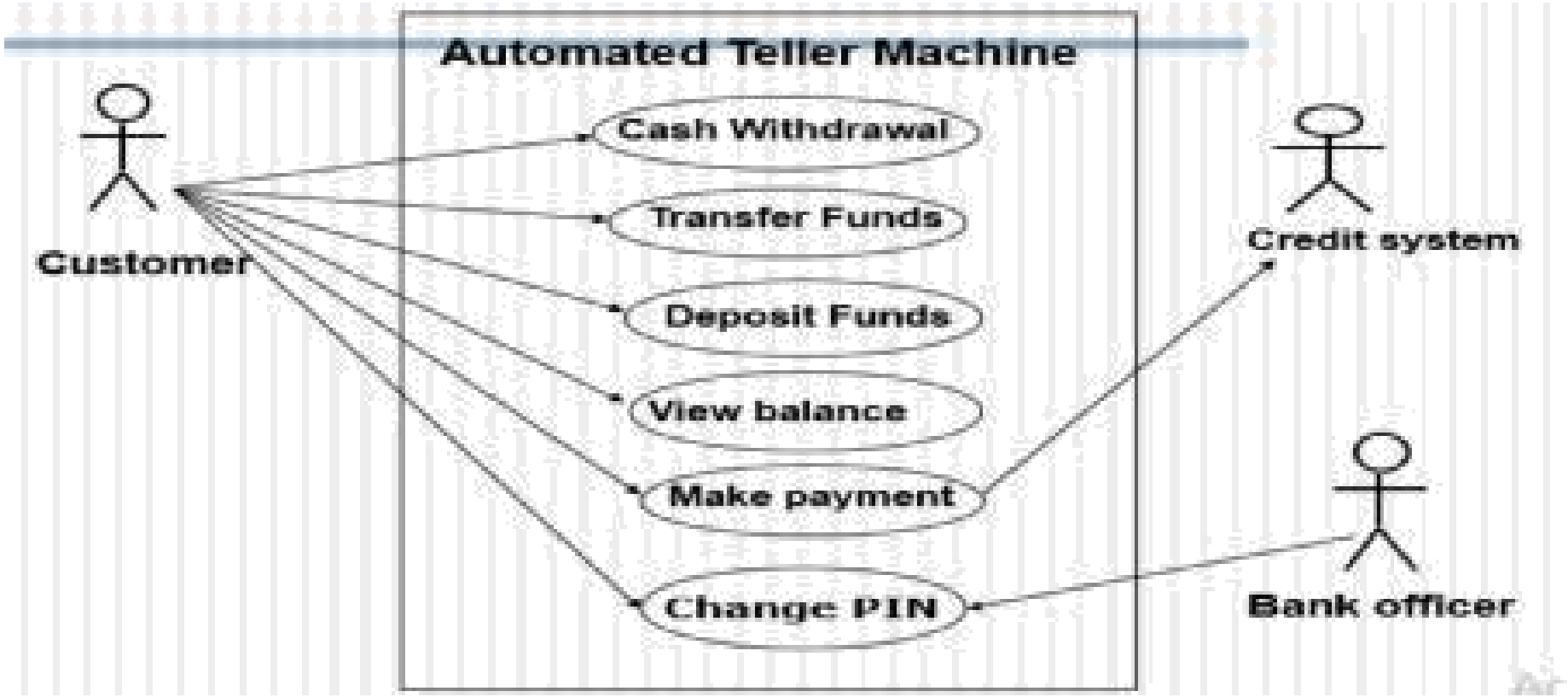
- A **use case** shows the behavior or functionality of a system (see Figure given below).
- It consists of a set of possible sequences of interactions between a system and a user in a particular environment, possible sequences that are related to a particular goal.
- A use case describes the behavior of a system under various conditions as the system responds to requests from principal actors.
- A principal actor initiates a request of the system, related to a goal, and the system responds.
- A use case can be stated as a present-tense verb phrase, containing the verb (what the system is supposed to do) and the object of the verb (what the system is to act on).
- For example, use case names would include Enter Sales Data, Compute Commission, Generate Quarterly Report.
- As with DFDs, use cases do not reflect all of the system requirements; they must be augmented by documents that detail requirements, such as business rules, data fields and formats, and complex formulas.



Definitions And Symbols Used In Use Cases

- Use case diagramming is relatively simple because it involves only a few symbols. However, like DFDs and other relatively simple diagramming tools, these few symbols can be used to represent quite complex situations.
- Mastering use case diagramming takes a lot of practice. The key symbols in a use case diagram are illustrated in Figure above and explained below:
  - *Actor:* As explained earlier, an actor is a role, not an individual. Individuals are instances of actors. One particular individual may play many roles simultaneously. An actor is involved with the functioning of a system at some basic level. **Actors are represented by stick figures**.
  - *Use case:* Each **use case is represented as an ellipse**. Each use case represents a single system function. The name of the use case can be listed inside the ellipse or just below it.
  - *System boundary:* The system boundary is represented as a box that includes all of the relevant use cases. Note that actors are outside the system boundary.
  - *Connections.* In Figure, note that the actors are connected to use cases with lines, and that use cases are connected to each other with arrows. A solid line connecting an actor to a use case shows that the actor is involved in that particular system function.
  - The solid line does not mean that the actor is sending data to or receiving data from the use case. Note that all of the actors in a use case diagram are not involved in all the use cases in the system.

- The dotted-line arrows that connect use cases also have labels (there is an <<extend>> label on the arrows in Figure). These use case connections and their labels are explained next. Note that use cases do not have to be connected to other use cases. The arrows between use cases do not illustrate data or process flows.
- Extend relationship. An extend relationship extends a use case by adding new behaviors or actions. It is shown as a dotted-line arrow pointing toward the use case that has been extended and labeled with the <<extend>> symbol.



UNIT I : “B” Origin of Software”

- **System Acquisition**
- Although there will always be some debate about when and where the first administrative information system was developed, there is general agreement that the first such system in the United Kingdom was developed at J. Lyons & Sons.
- In the United States, the first administrative information system was General Electric’s (GE) payroll system, which was developed in 1954 (Computer History Museum, 2003). At that time, and for many years afterward, obtaining an information system meant one thing only: in- house development.
- The software industry did not even come into existence until a decade after GE’s payroll system was implemented.
- Since GE’s payroll system was built, in-house development has become a progressively smaller piece of all the systems development work that takes place in and for organizations.
- Internal corporate information systems departments now spend a smaller and smaller proportion of their time and effort on developing systems from scratch.
- Companies continue to spend relatively little time and money on traditional software development and maintenance.
- Instead, they invest in packaged software, open-source software, and outsourced services. Organizations today have many choices when seeking an information system. We will start with a discussion of outsourcing development and operation and then move on to a presentation on the sources of software.

Outsourcing

- If one organization develops or runs a computer application for another organization, that practice is called **outsourcing**.
- Outsourcing includes a spectrum of working arrangements. At one extreme is having a firm develop and run your application on its computers—all you do is supply input and take output.
- A common example of such an arrangement is a company that runs payroll applications for clients so that clients do not have to develop an independent in-house payroll system.
- Instead, they simply provide employee payroll information to the company, and, for a fee, the company returns completed paychecks, payroll accounting reports, and tax and other statements for employees. For many organizations, payroll is a very cost-effective operation when outsourced in this way.
- Another example of outsourcing would be if you hired a company to run your applications at your site on your computers. In some cases, an organization employing such an arrangement will dissolve some or all of its information systems (IS) unit and fire all of its IS employees. Often the company brought in to run the organization's computing will rehire many of the organization's original IS unit employees.
- **The reasons behind outsourcing are**:
- freeing up internal resources,
- increasing the revenue potential of the organization,
- reducing time to market,
- increasing process efficiencies, and
- outsourcing noncore activities.

Sources of Software

- We can group the sources of software into six major categories: information technology services firms, packaged software producers, enterprise-wide solutions, cloud computing vendors, open-source software, and in-house developers (Figure given below). These various sources represent points along a continuum of options, with many hybrid combinations along the way.

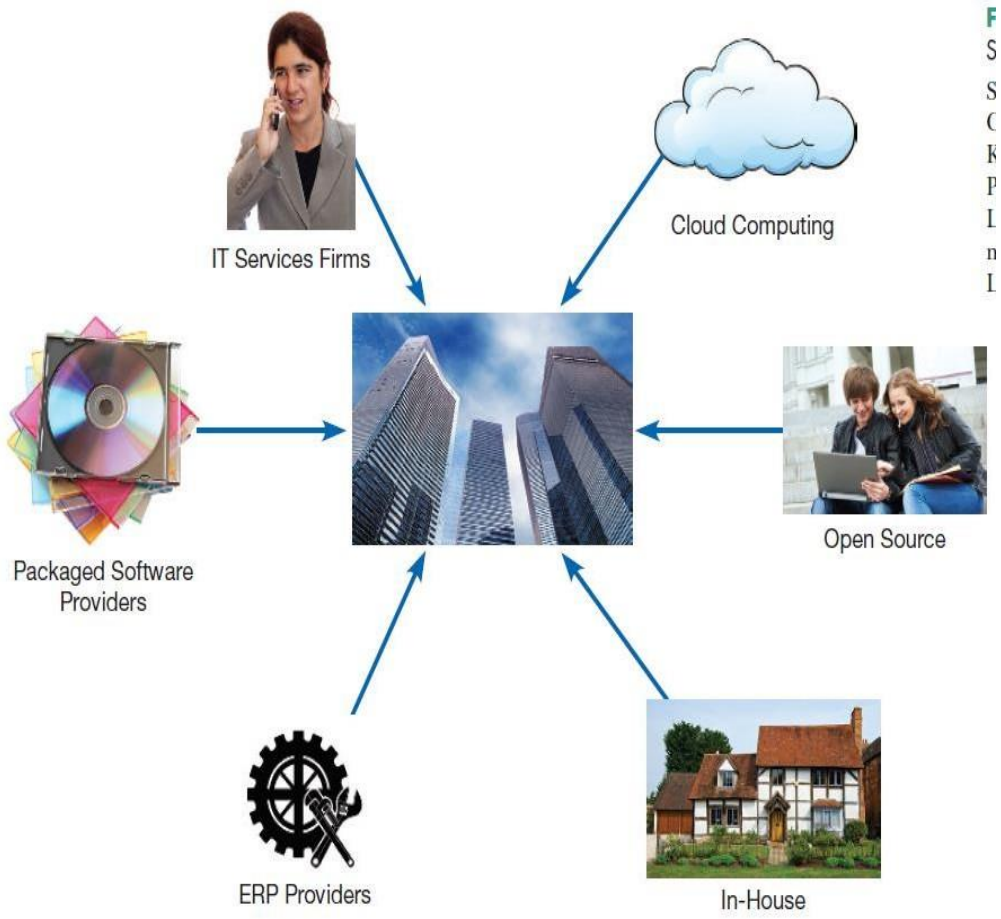


TABLE 2-1 Leading Software Firms and Their Development Specializations

Specialization	Example Firms or Websites
IT Services	Accenture Computer Sciences Corporation (CSC) IBM HP
Packaged Software Providers	Intuit Microsoft Oracle SAP AG Symantec
Enterprise Software Solutions	Oracle SAP AG
Cloud Computing	Amazon.com Google IBM Microsoft Salesforce.com
Open Source	SourceForge.net

TABLE 2-2 Comparison of Six Different Sources of Software Components

Producers	When to Go to This Type of Organization for Software	Internal Staffing Requirements
IT services firms	When task requires custom support and system can't be built internally or system needs to be sourced	Internal staff may be needed, depending on application
Packaged software producers	When supported task is generic	Some IS and user staff to define requirements and evaluate packages
Enterprise-wide solutions vendors	For complete systems that cross functional boundaries	Some internal staff necessary but mostly need consultants
Cloud computing	For instant access to an application; when supported task is generic	Few; frees up staff for other IT work
Open-source software	When supported task is generic but cost is an issue	Some IS and user staff to define requirements and evaluate packages
In-house developers	When resources and staff are available and system must be built from scratch	Internal staff necessary though staff size may vary

UNIT I : “C” Managing the Information System Project

- The focus of this chapter was on managing information systems projects and the role of the project manager in this process. A project manager has both technical and managerial skills and is ultimately responsible for determining the size, scope, and resource requirements for a project.
- Once a project is deemed feasible by an organization, the project manager ensures that the project meets the customer’s needs and is delivered within budget and time constraints. To manage the project, the project manager must execute four primary activities: project initiation, project planning, project execution, and project closedown. The focus of project initiation is on assessing the size, scope, and complexity of a project and on establishing procedures to support later project activities.
- The focus of project planning is on defining clear, discrete activities and the work needed to complete each activity. The focus of project execution is on putting the plans developed in project initiation and planning into action. Project closedown focuses on bringing the project to an end.
- Gantt charts and network diagrams are powerful graphical techniques used in planning and controlling projects.
- Both Gantt charts and network diagram scheduling techniques require that a project have activities that can be defined as having a clear beginning and end, can be worked on independently of other activities, are ordered, and are such that their completion signifies the end of the project.
- Gantt charts use horizontal bars to represent the beginning, duration, and ending of an activity.

Contd…

- Network diagramming is a critical path scheduling method that shows the interrelationships among activities.
- Critical path scheduling refers to planning methods whereby the order and duration of the project's activities directly affect the completion date of the project.
- These charts show when activities can begin and end, which activities cannot be delayed without delaying the whole project, how much slack time each activity has, and progress against planned activities.
- A network diagram's ability to use probability estimates in determining critical paths and deadlines makes it a widely used technique for very complex projects.
- A wide variety of automated tools for assisting the project manager is available.
- Most tools have a set of common features, including the ability to define and order tasks, assign resources to tasks, and modify tasks and resources. Systems vary regarding the number of activities supported, the complexity of relationships, processing and storage requirements, and cost.

Assignments

1. Define the system, and List its various characteristics.
2. When would you use the agile methodologies. How is it different from waterfall approach system development.
3. With proper reasoning, explain how CASE Tools aid in information system development? You have been hired as a system analyst in TU tech software development company and you are asked to analyze the way system works. What qualities do you need to have to analyze such type of systems?
4. List of OOAD. Differentiate between structured methodologies and object oriented methodologies.

MCQ

i) Which of the following Information systems are aimed at improving the routine business activities on which all organizations depend?

a)Management Information systems
b) Decision support systems
c) Transaction Processing Systems
d) Executive Information

ii) The project life cycle consists of

a) Understanding the scope of the project
b) Objectives of the project
c) Formulation and planning various activities
d) All of the above

iii) Which is the most important feature of spiral model?

a) Quality Management
b) Efficiency Management
c) Risk Management
d) Performance Management

iv) …… includes the existing system, the proposed system, system flow charts, modular design of the system, print layout charts and data file designs.

a)Feasibility Report b) Functional Specification Report
c) Design Specification Report d) Terms of Reference

v) For the best Software model suitable for the project, in which of the phase the developers decide a roadmap for project plan?

a) Software b) System Analysis
c) Coding d) Testing