OO Software Engineering

- Nature of the course: theory+ Case Studies+project
- Credit Hours: 3

Syllabus

- Unit 1: 4 Hrs Software life cycle models, Requirement analysis and specification, object oriented software development,
- Unit 2: 5 Hrs Introduction to object-orientation, object- oriented system development-function /data methods, object oriented analysis, construction and testing, object -oriented programming with examples

Syllabus

- Unit 3: 10 Hrs Architecture-model architecture, requirements, analysis, design, implementation and test model, analysis, construction, Real-time-classification of real time systems, database-RDBMS, Object DBMS, components-use of components, component management, testing-on testing, Unit, integration, system and the testing process

Syllabus

- Unit 4: 6 Hrs Managing object-oriented software engineering, Project selection and preparation, product development organization, project organization and management, project staffing, software quality assurance, software metrics
- Unit 5: 5 Hrs Object oriented analysis and design, hierarchical object-Oriented design, object Modeling technique and responsibility-driven design
- Unit 6: 15 Hrs Case studies and project Warehouse management system, Telecom

Software Engineering

Reference Books

- Ivar Jacobson-Object Oriented Software Engineering
- IAN SOMMERVILLE, Software Engineering
- Grady Booch-Object-oriented analysis and design

Software Engineering

Reference Books

- IAN SOMMERVILLE, Software Engineering
- ROGER S. PRESSMAN, Software Engineering a practitioner’s Approach

Is there an "official" name?

Software Engineering! *Information Systems Engineering!*

Systems Engineering! *Information Systems Development!*

Introduction to Software engineering

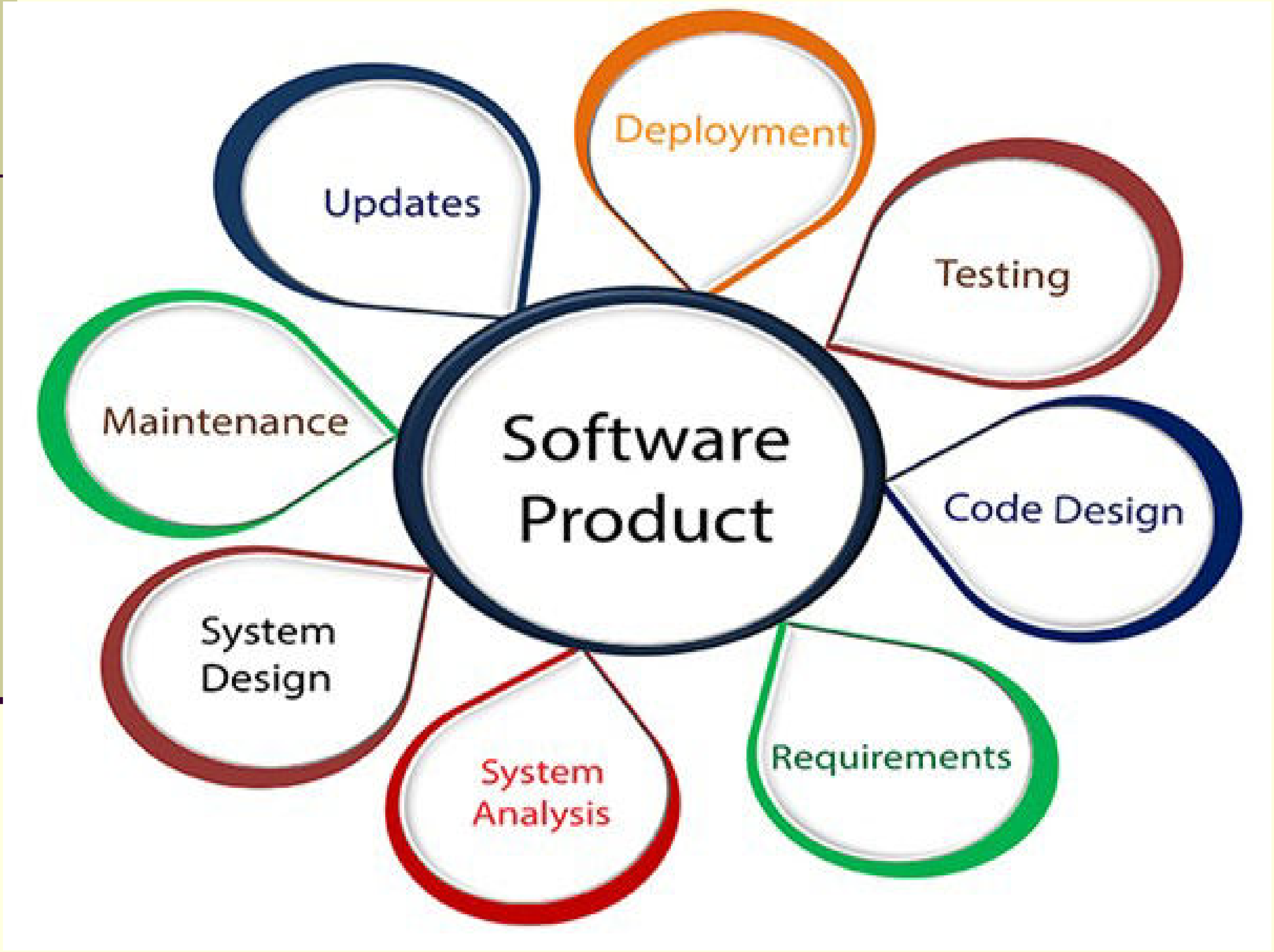
- The economies of ALL developed nations are dependent on software
- More and more systems are software controlled
- Software engineering is concerned with theories, methods and tools for professional software development

Software costs

- Software costs often dominate system costs. The costs of software on a PC are often greater than the hardware cost
- Software costs more to maintain than it does to develop.
- For systems with a long life, maintenance costs may be several times development costs
- Software engineering is concerned with cost-effective software development

Software

- Computer programs and associated documentation
- Software products may be developed for a particular customer or may be developed for a general market
- Software products may be
  - Generic - developed to be sold to a range of different customers
  - Custom - developed for a single customer according to their specification


Software Product
Deployment
Testing
Code Design
Requirements
System Analysis
System Design
Maintenance
Updates

Software Engineering

- Software engineering is an **engineering** discipline which is concerned with all aspects of software production
- Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available

Difference between software engineering and computer science

- Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software
- Computer science theories are currently insufficient to act as a complete form the basis for software engineering

Successful software system

- Software development projects have not always been successful
- When do we consider a software application successful?
  - Development completed
  - It is useful
  - It is usable, and
  - It is used
- Cost-effectiveness, maintainability implied

Reasons for failure

- Schedule slippage
- Cost over-runs
- Does not solve user’s problem
- Poor quality of software
- Poor maintainability

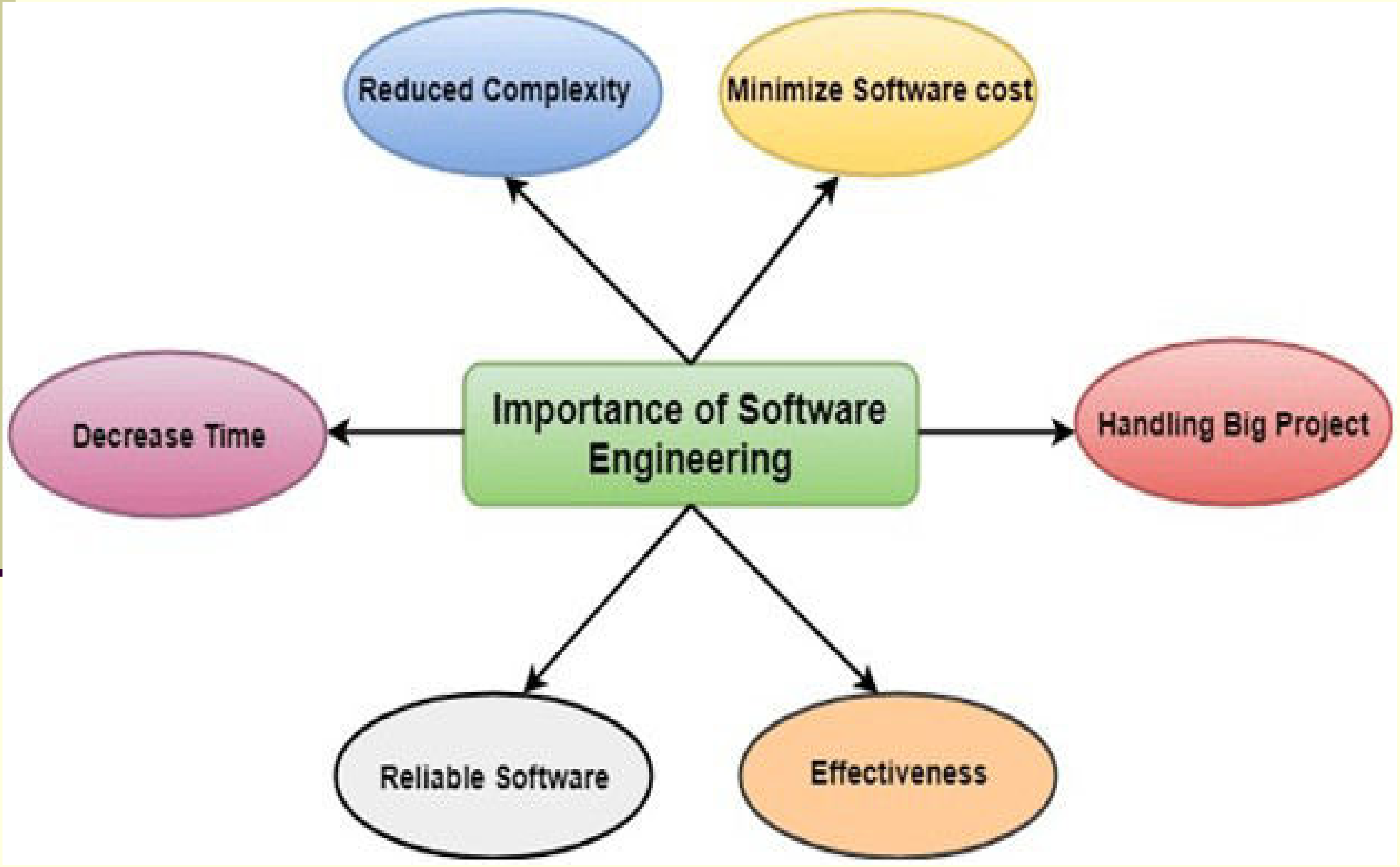
Reasons for failure ….

- Ad hoc software development results in such problems
  - No planning of development work (e.g. no milestones defined)
  - Deliverables to user not identified
  - Poor understanding of user requirements
  - No control or review
  - Technical incompetence of developers
  - Poor understanding of cost and effort by both developer and user

Software projects

- Software is different from other products
  - Cost of production concentrated in development
  - Maintenance consists of making corrections and enhancing or adding functions
  - Progress in development is difficult to measure

Importance of Software Engineering



Apply Engineering Approach

- Hence planning and control even more important in software development ⇨ engineering approach:
  - Attempt to estimate cost/effort
  - Plan and schedule work
  - Involve user in defining requirements
  - Identify stages in development
  - Define clear milestones so that progress can be measured
  - Schedule reviews both for control and quality
  - Define deliverables
  - Plan extensive testing

Job of Software Developer is difficult

- Dealing with users
  - defined requirements
  - Concern with ease-of-use and response time
- Dealing with technical people
  - Concerned with coding, databases, file structures, etc.
- Dealing with management
  - Concerned with return on their investment
  - Cost-benefit analysis
  - Schedule

Software Process

- Process consists of activities/steps to be carried out in a particular order
- Software process deals with both technical and management issues
- Consists of different types of process
- Process for software development: produces software as end-result
  - multiple such processes may exist
  - a project follows a particular process

Process Types …

- Process for managing the project
  - defines project planning and control
  - effort estimations made and schedule prepared
  - resources are provided
  - feedback taken for quality assurance
  - monitoring done.

Process Types …

- Process for change and configuration mgmt.
  - Resolving requests for changes
  - Defining versions, their compositions
  - Release control
- Process for managing the above processes themselves
  - Improving the processes based on new techniques, tools, etc.
  - Standardizations and certifications (ISO, CMM) , (The Capability Maturity Model (CMM) is a methodology used to develop and refine an organization's software development process)

Multiple processes

- A large software development company may have multiple development processes
  A. For client-server based conventional applications (sales processing, payroll)
  B. For enterprise-level (ERP) projects based on packages and customization
  C. For web-based e-commerce type
  D. For data-warehousing/decision-support type
- The company may have many projects in each category

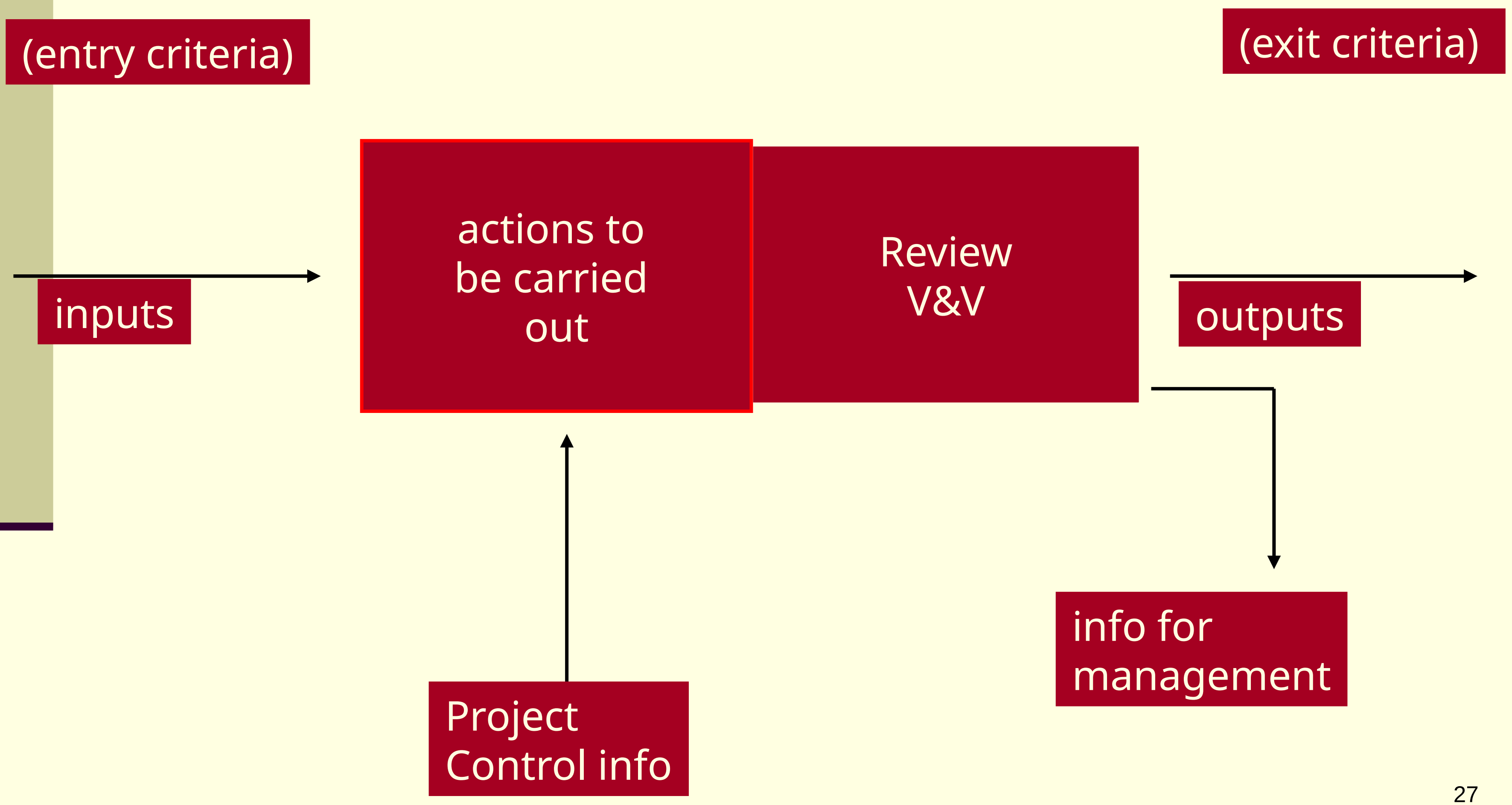
Step in a Process

- Each step has a well-defined objective
- Requires people with specific skills
- Takes specific inputs and produces well-defined outputs
- Step defines when it may begin (entry criteria) and when it ends (exit criteria)
- Uses specific techniques, tools, guidelines, conventions.

Process step …

- Step must be executed as per project plan that gives duration, effort, resources, constraints, etc.
- It must produce information for management so that corrective actions can be taken
  - E.g., adding more resources
- A step ends in a review (V&V)
  - Verification: check consistency of outputs with inputs (of the step)
  - Validation: check consistency with user needs

Process step



What is a software process?

- A set of activities whose goal is the development or evolution of software
- Generic activities in all software processes are:
  - Specification - what the system should do and its development constraints
  - Development - production of the software system
  - Validation - checking that the software is what the customer wants
  - Evolution - changing the software in response to changing demands

What is a software process model?

- Generic process models
  - Waterfall model (specification, development, validation, evolution and represents them as separate process phases)
  - Evolutionary development (specification, development, validation)
  - Formal transformation or formal systems development (Using Mathematical methods to construct a program)
  - Integration from reusable components or reuse-based development

What are the costs of software engineering?

- Roughly 60% of costs are development costs, 40% are testing costs.
- For custom software, evolution costs often exceed development costs
- Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability
- Distribution of costs depends on the development model that is used

CASE (Computer-Aided Software Engineering)

- Software systems which are intended to provide automated support for software process activities. CASE systems are often used for method support
- Upper-CASE
  - Tools to support the early process activities of requirements and design
- Lower-CASE
  - Tools to support later activities such as programming, debugging and testing

Attributes of good software

- The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable
- Maintainability
  - Software must evolve to meet changing needs
- Dependability
  - Software must be trustworthy
- Efficiency
  - Software should not make wasteful use of system resources
- Usability
  - Software must be usable by the users for which it was designed

Key challenges facing software engineering

- Coping with legacy systems, coping with increasing diversity and coping with demands for reduced delivery times
- Legacy systems
  - Old, valuable systems must be maintained and updated
- Heterogeneity
  - Systems are distributed and include a mix of hardware and software
- Delivery
  - There is increasing pressure for faster delivery of software

Professional and ethical responsibility

- Software engineering involves wider responsibilities than simply the application of technical skills
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals
- Ethical behaviour is more than simply upholding the law.

Issues of professional responsibility

- *Confidentiality*
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- *Competence*
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outside their competence.

Issues of professional responsibility

- *Intellectual property rights*
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- *Computer misuse*
  - Software engineers should not use their technical skills to misuse other people’s computers.

Summary

- Software engineering is an engineering discipline which is concerned with all aspects of software production.
- Software products consist of developed programs and associated documentation. Essential product attributes are maintainability, dependability, efficiency and usability.
- The software process consists of activities which are involved in developing software products. Basic activities are software specification, development, validation and evolution.
- Methods are organised ways of producing software. They include suggestions for the process to be followed, the notations to be used, rules governing the system descriptions which are produced and design guidelines.

Summary

- CASE tools are software systems which are designed to support routine activities in the software process such as editing design diagrams, checking diagram consistency and keeping track of program tests which have been run.
- Software engineers have responsibilities to the engineering profession and society. They should not simply be concerned with technical issues.
- Professional societies publish codes of conduct which set out the standards of behaviour expected of their members.

Questions

- What is the prime objective of software engineering?
- Give the importance of software engineering
- Can you differentiate computer software and computer program?
- What is software scope?
- What are CASE tools?
- What are the various categories of software?
- What are the challenges in software?
- What is a software process?

Software life cycle models

Software Processes

- Coherent sets of activities for specifying, designing, implementing and testing software systems

Objectives

- To introduce software process models
- To describe a number of different process models and when they may be used
- To describe process models for requirements engineering, software development, testing and evolution

The software process

- A structured set of activities required to develop a software system
  - Specification
  - Design
  - Validation
  - Evolution
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective

Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development
- Prototyping Model
- Evolutionary development
  - Specification and development are interleaved
- Reuse-based development
  - The system is assembled from existing components

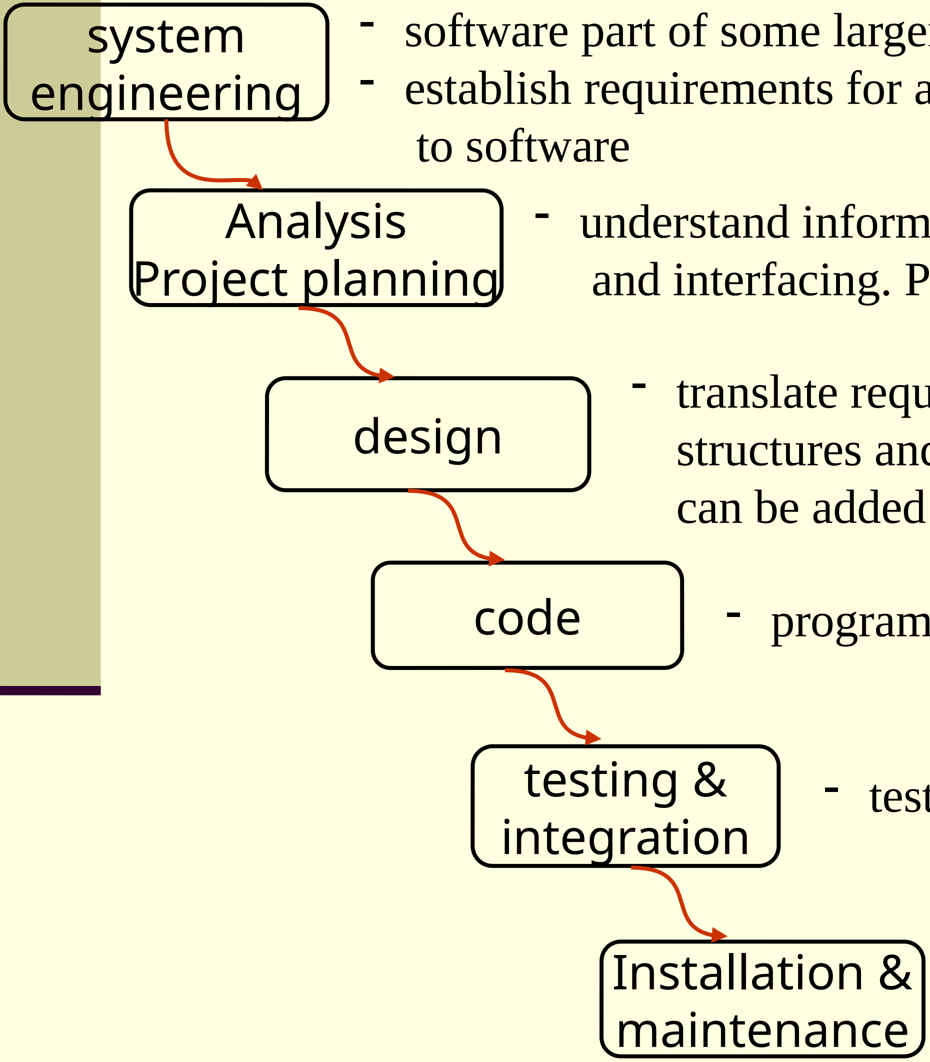
Waterfall Model for Development

- Here, steps (phases) are arranged in linear order
  - A step take inputs from previous step, gives output to next step (if any)
  - Exit criteria of a step must match with entry criteria of the succeeding step
- It follows ‘specify, design, build’ sequence that is intuitively obvious and appears natural

Waterfall Model …

- Produces many intermediate deliverables, usually documents
  - Standard formats defined
  - Act as ‘baseline’ used as reference (for contractual obligations, for maintenance)
  - Important for quality assurance, project management, etc.
- It is widely used (with minor variations) when requirements are well understood

Waterfall Model



- software part of some larger system
- establish requirements for all elements of the system; assign some to software

- understand information domain, functions, performance and interfacing. Project plans made

- translate requirements into s/w architecture, data structures and procedural details. A ‘detailed design’ step can be added

- programming

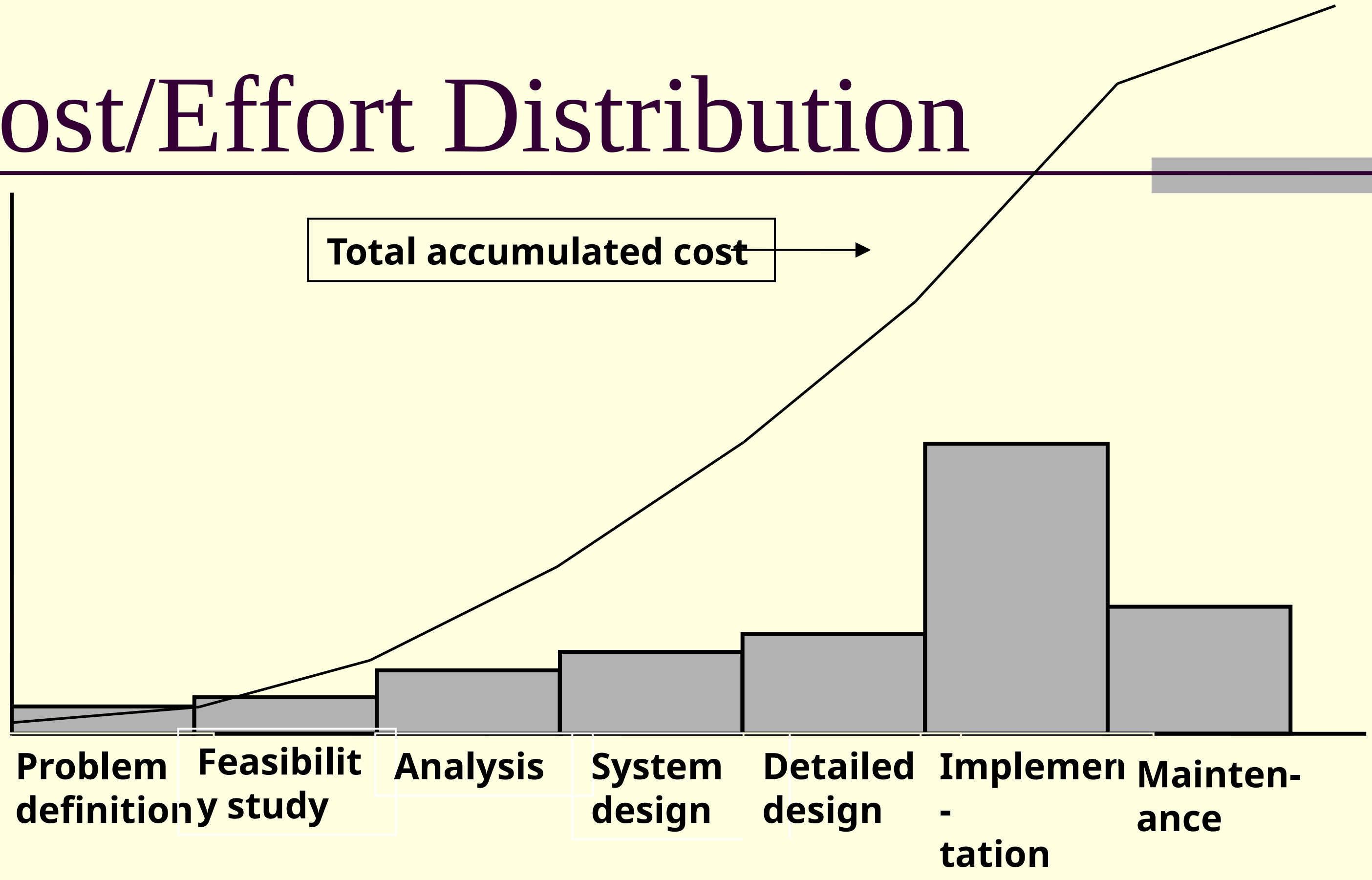
- test logic and function interfaces

- deployment; make changes for
  - Errors, performance
  - changes in requirement

Deliverables in Waterfall Model

- Project plan and feasibility report
- Requirements document (SRS : Software Requirement Specifications)
- System design document
- Detailed design document
- Test plans and test reports
- Source code
- Software manuals (user manual, installation manual)
- Review reports

Cost/Effort Distribution



- Accumulated cost increases dramatically as programmers, operators, technical writers and computer time is committed.
- Cost of discovering and fixing errors also increases with steps

Shortcomings of Waterfall Model

- Requirements may not be clearly known, especially for applications not having existing (manual) counterpart
  - Railway reservation: manual system existes, so SRS can be defined
  - On-line container management for railways - new
  - Knowledge management for a central bank – new

Shortcomings …

- Requirements change with time during project life cycle itself
  - Users may find solution of little use
  - Better to develop in parts in smaller increments; this is common for packages, system software
- Considered documentation-heavy: so much documentation may not be required for all types of projects.

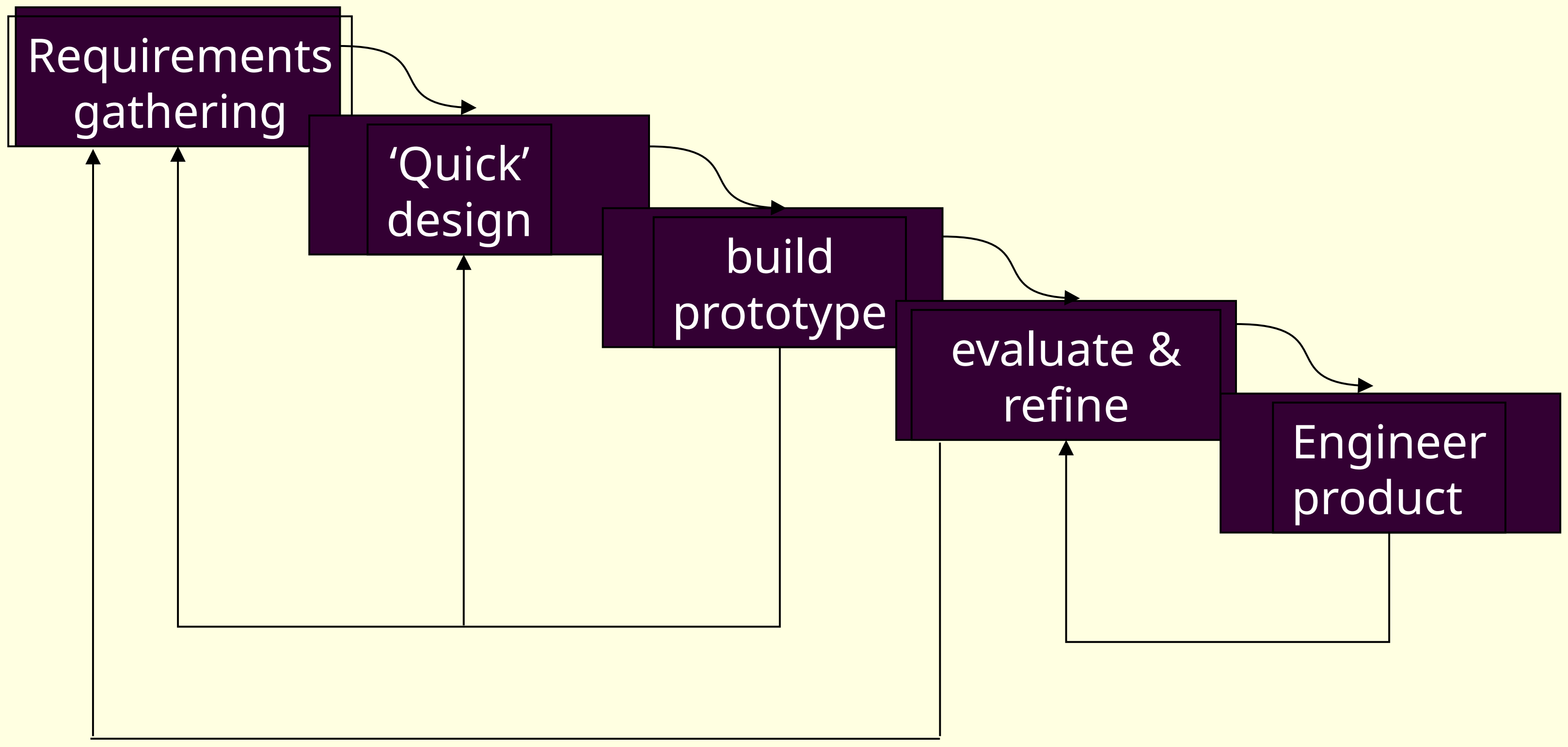
Prototyping Model

- When customer or developer is not sure
  - Of requirements (inputs, outputs)
  - Of algorithms, efficiency, human-machine interaction
- A throwaway prototype built from currently known user needs
- Working or even ‘paper’ prototype

Prototyping …

- Quick design focuses on aspects visible to user; features clearly understood need not be implemented
- Prototype is tuned to satisfy customer needs
  - Many iterations may be required to incorporate changes and new requirements
- Final product follows usual define-design-build-test life cycle

Prototyping



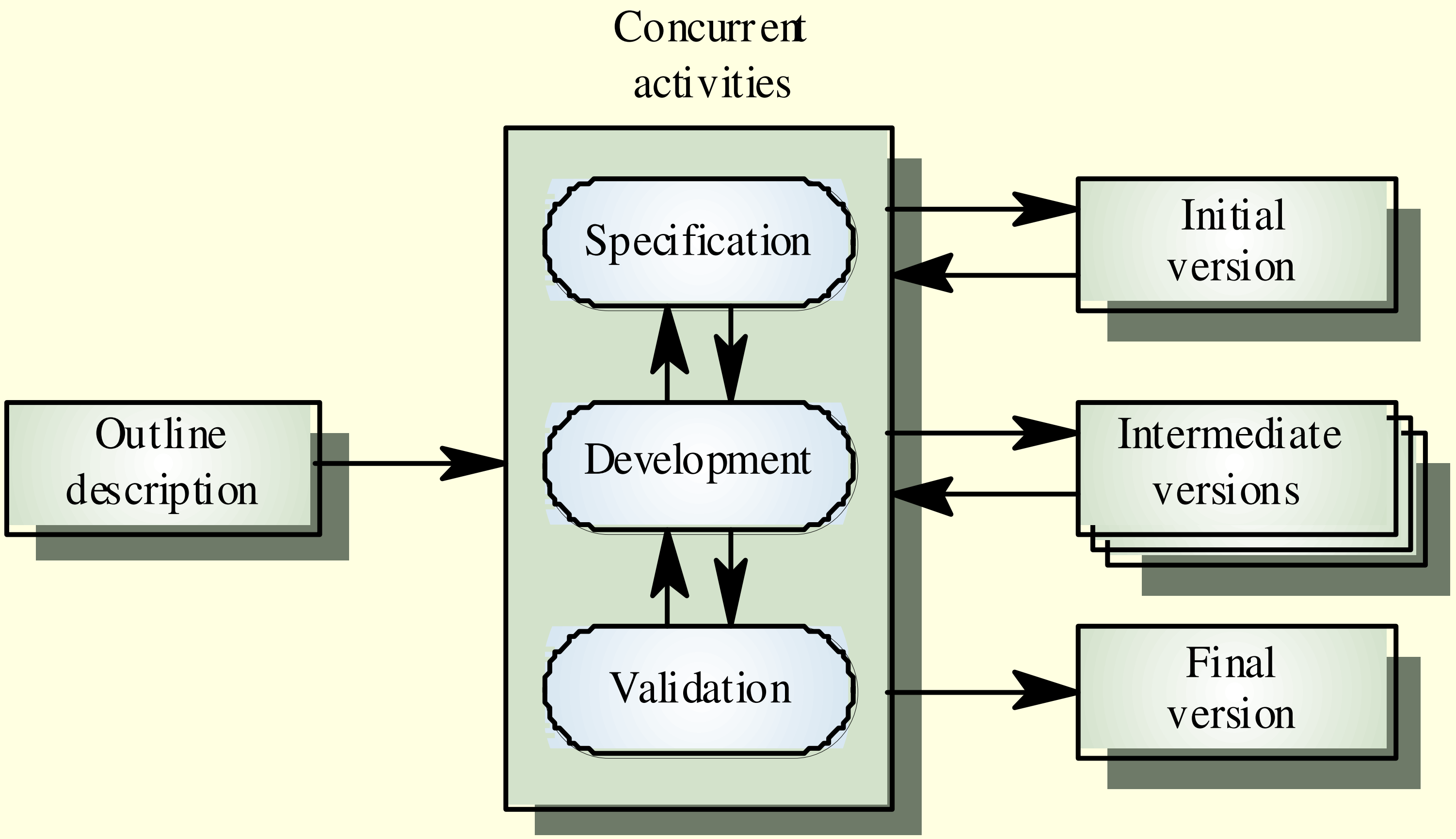
Limitations of Prototyping

- Customer may want prototype itself !
- Developer may continue with implementation choices made during prototyping
  - may not give required quality, performance, ….
- Good tools need to be acquired for quick development
- May increase project cost

Iterative Development

- Useful for product development where developers define scope, features to serve many customers
- Early version with limited feature important to establish market and get customer feedback
- Initial version may follow any method
- A list of features for future versions maintained
- Each version is analyzed to decide feature list for next iteration

- **Evolutionary Development model [SE-7, Fig 4.2]**



Evolutionary Development..

- Main characteristics:
  - The phases of the software construction are interleaved
  - Feedback from the user is used throughout the entire process
  - The software product is refined through many versions
- Types of evolutionary development:
  - Exploratory development
  - Throw-away prototyping

Software Development Life Cycle

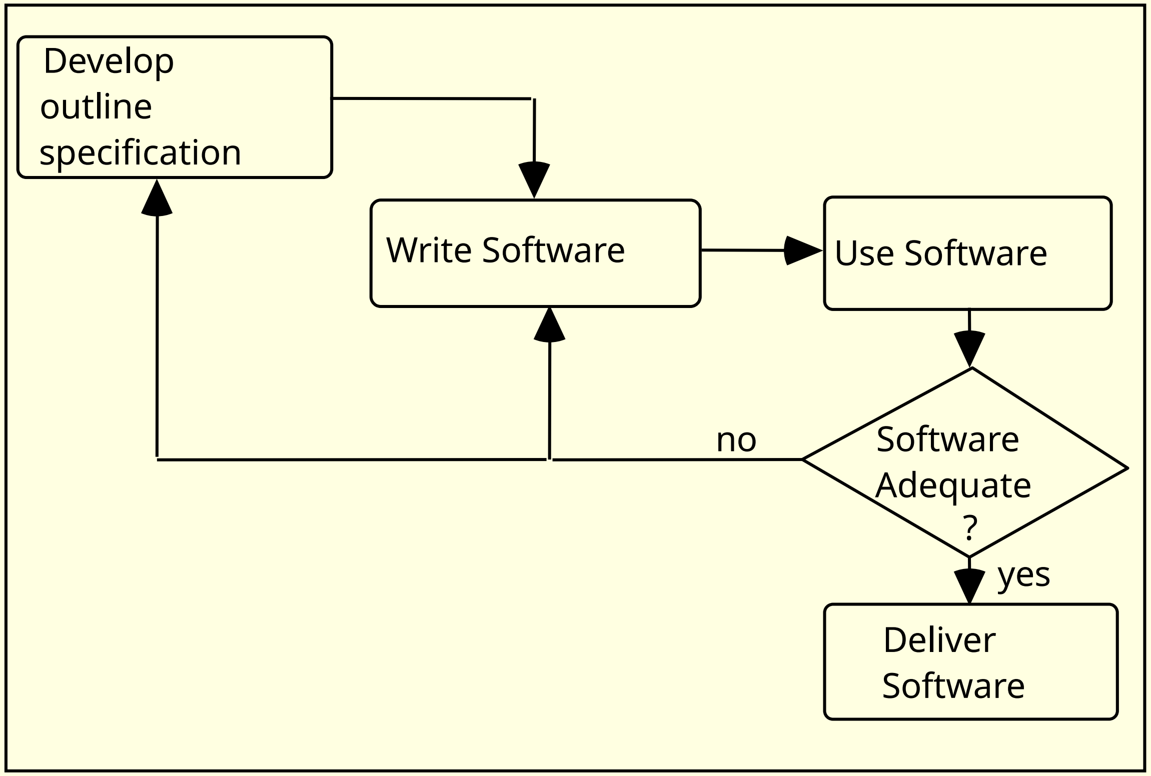


Figure 2: exploratory development

Softawre Development Life Cycle

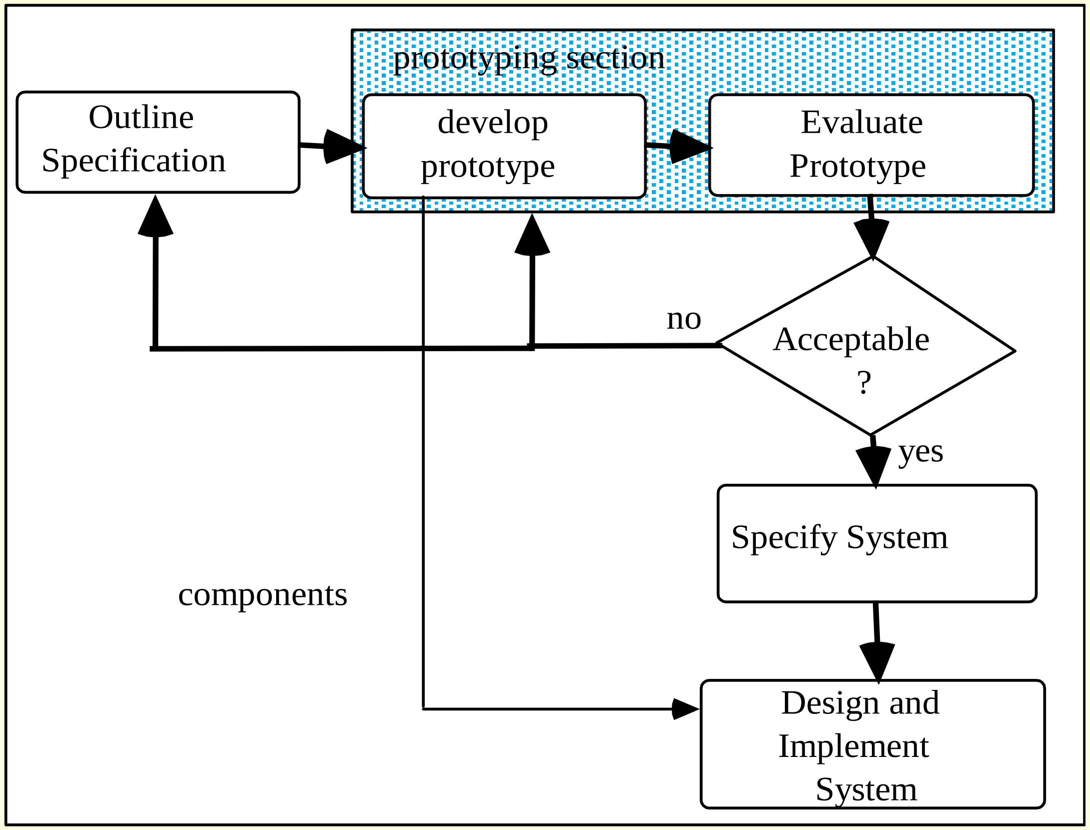


Figure 1: prototyping within software life cycle

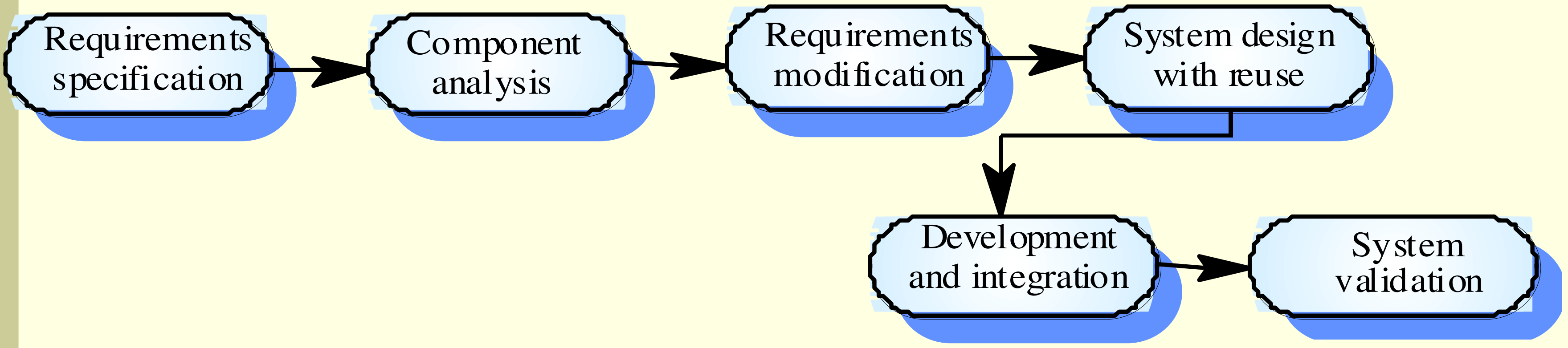
Evolutionary Development…

- **Advantages:**
  - **Deals constantly with changes**
  - **Provides quickly an initial version of the system**
  - **Involves all development teams**
- **Disadvantages:**
  - **Quick fixes may be involved**
  - **“Invisible” process, not well-supported by documentation**
  - **The system’s structure can be corrupted by continuous change**

Evolutionary Development …

- Disadvantages [cont'd]:
  - Special tools and techniques may be necessary
  - The client may have the impression the first version is very close to the final product and thus be less patient
- Applicability:
  - When requirements are not well understood
  - When the client and the developer agree on a "rapid prototype" that will be thrown away
  - Good for small and medium-sized software systems

Component-based Software Engineering


Requirements specification
Component analysis
Requirements modification
System design with reuse
Development and integration
System validation

Component-based Software Engineering..

- Main characteristics:
  - Makes intensive use of existing reusable components
  - The focus is on integrating the components rather than on creating them from the scratch

Component-based Software Engineering.

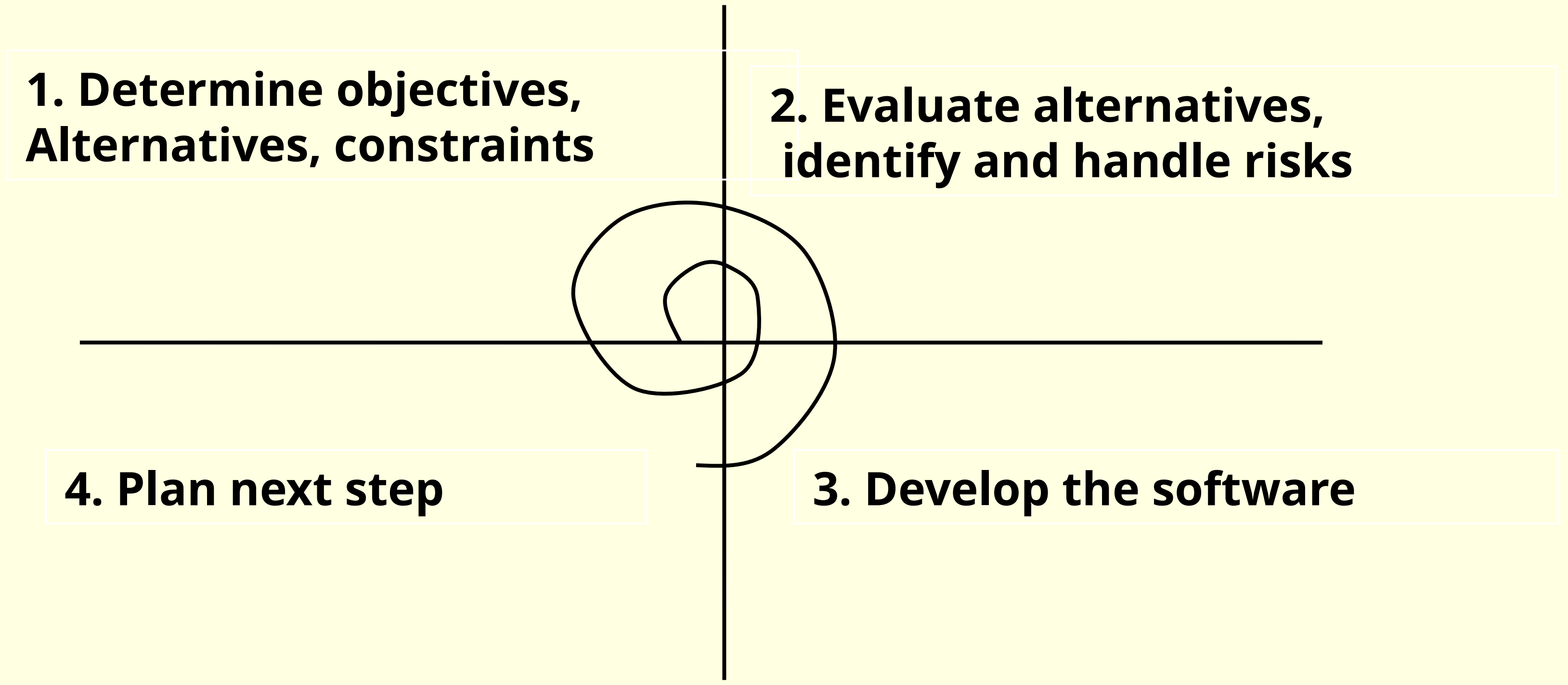
- Advantages:
  - Reduces considerably the software to be developed “in-house”
  - Allows faster delivery
  - In principle, more reliable systems, due to using previously tested components

Component-based Software Engineering

- Disadvantages:
  - Compromises in requirements are needed
  - Less control over the system’s evolution
- Applicability:
  - When there is a pool of existing components that could satisfy the requirements of the new product
  - Emerging trend: integration of web services from a range of suppliers

Spiral Model

- Activities are organized in a spiral having many cycles
- Four quadrants in each cycle


1. Determine objectives,
Alternatives, constraints
2. Evaluate alternatives,
identify and handle risks
4. Plan next step
3. Develop the software

Spiral Model …

- Prototyping, simulations, benchmarking may be done to resolve uncertainties/risks
- Development step depends on remaining risks; e.g.,
  - Do prototype for user interface risks
  - Use basic waterfall model when user interface and performance issues are understood but only development risk remains
- Risk driven : allows us mix of specification-oriented, prototype-oriented, simulation based or any other approach.

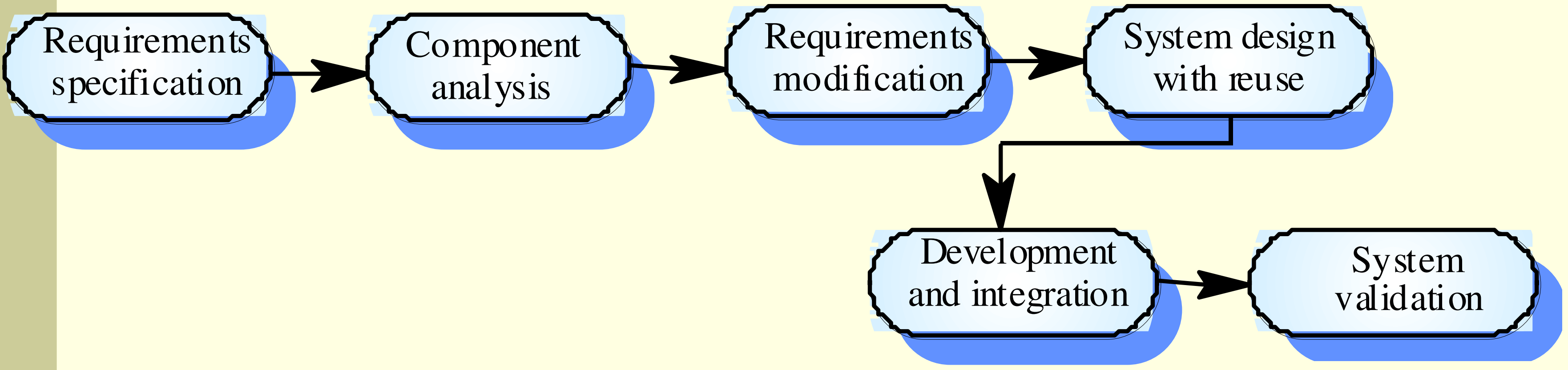
Spiral Model

- Main characteristics:
  - Also a hybrid model that support process iteration
  - The process is represented as a spiral, each loop in the spiral representing a process phase
  - Four sectors per loop: objective setting, risk assessment and reduction, development and validation, planning
  - Risk is explicitly taken into consideration

Spiral Model

- Advantages:
  - Risk reduction mechanisms are in place
  - Supports iteration and reflects real-world practices
  - Systematic approach
- Disadvantages:
  - Requires expertise in risk evaluation and reduction
  - Complex, relatively difficult to follow strictly
  - Applicable only to large systems
- Applicability:
  - Internal development of large systems

Reuse-oriented development


Requirements specification
Component analysis
Requirements modification
System design with reuse
Development and integration
System validation

Comparison of Different Life Cycle Models

- **Iterative waterfall model**
  - most widely used model.
  - But, suitable only for well-understood problems**.**
- **Prototype model is suitable for projects not well understood:**
  - **user requirements**
  - **technical aspects**

Comparison of Different Life Cycle Models (CONT.)

- **Evolutionary model is suitable for large problems:**
  - can be decomposed into a set of modules that can be incrementally implemented,
  - incremental delivery of the system is acceptable to the customer.
- **The spiral model:**
  - suitable for development of technically challenging software products that are subject to several kinds of risks.

Summary

- Software processes are the activities involved in producing and evolving a software system. They are represented in a software process model
- General activities are specification, design and implementation, validation and evolution
- Generic process models describe the organisation of software processes
- Iterative process models describe the software process as a cycle of activities

Summary

- Design and implementation processes transform the specification to an executable program
- Validation involves checking that the system meets to its specification and user needs
- Evolution is concerned with modifying the system after it is in use
- CASE technology supports software process activities

Software metrics

Introduction

- Software metrics in software engineering are the standards for estimating the quality, **progress**, and health of **software development** activity.
- A software metric is a quantifiable or countable assessment of the qualities of software.

Introduction

- A software metric is a measure of software characteristics which are measurable or countable.
- Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.
- Within the software development process, many metrics are that are all connected.

Software Metrics

- A metric is a measurement of the level at which any impute belongs to a system product or process.
- Software metrics are a quantifiable or countable assessment of the attributes of a software product. There are 4 functions related to software metrics:
- **Planning**
- **Organizing**
- **Controlling**
- **Improving**

Characteristics of software Metrics

- **Quantitative:** Metrics must possess a quantitative nature. It means metrics can be expressed in numerical values.
- **Understandable:** Metric computation should be easily understood, and the method of computing metrics should be clearly defined.
- **Applicability:** Metrics should be applicable in the initial phases of the development of the software.
- **Repeatable:** When measured repeatedly, the metric values should be the same and consistent.
- **Economical:** The computation of metrics should be economical.
- **Language Independent:** Metrics should not depend on any programming language.

Types of Software Metrics

- **Product Metrics:** Product metrics are used to evaluate the state of the product, tracing risks and undercover prospective problem areas. The ability of the team to control quality is evaluated. Examples include lines of code, cyclomatic complexity, code coverage, defect density, and code maintainability index.
- **Process Metrics:** Process metrics pay particular attention to enhancing the long-term process of the team or organization. These metrics are used to optimize the development process and maintenance activities of software. Examples include effort variance, schedule variance, defect injection rate, and lead time.

Types of Software Metrics

Project Metrics: The project metrics describes the characteristic and execution of a project.

Examples include effort estimation accuracy, schedule deviation, cost variance, and productivity.

Usually measures

- Number of software developer
- Staffing patterns over the life cycle of software
- Cost and schedule
- Productivity

Advantages of Software Metrics

- Reduction in cost or budget.
- It helps to identify the particular area for improvising.
- It helps to increase the product quality.
- Managing the workloads and teams.
- Reduction in overall time to produce the product,.
- It helps to determine the complexity of the code and to test the code with resources.
- It helps in providing effective planning, controlling and managing of the entire product.

Disadvantages of Software Metrics

- It is expensive and difficult to implement the metrics in some cases.
- Performance of the entire team or an individual from the team can’t be determined. Only the performance of the product is determined.
- Sometimes the quality of the product is not met with the expectation.
- It leads to measure the unwanted data which is wastage of time.
- Measuring the incorrect data leads to make wrong decision making.

Software Project Management

Software Project

- A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Need of software project management

- Software is said to be an intangible product. Software development is a kind of all new stream in world business and there’s very little experience in building software products.
- Most software products are tailor made to fit client’s requirements.
- The most important is that the underlying technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one.
- All such business and environmental constraints bring risk in software development hence it is essential to manage software projects efficiently.

Need of software project management

- The image above shows triple constraints for software projects.
- It is an essential part of software organization to deliver quality product, keeping the cost within client’s budget constrain and deliver the project as per scheduled.
- There are several factors, both internal and external, which may impact this triple constrain triangle.
- Any of three factor can severely impact the other two.
- Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

Software Project Manager

- A software project manager is a person who undertakes the responsibility of executing the software project.
- Software project manager is thoroughly aware of all the phases of SDLC that the software would go through.
- Project manager may never directly involve in producing the end product but he controls and manages the activities involved in production.
- A project manager closely monitors the development process, prepares and executes various plans, arranges necessary and adequate resources, maintains communication among all team members in order to address issues of cost, budget, resources, time, quality and customer satisfaction.

Managing People

- Act as project leader
- Liaison with stakeholders
- Managing human resources
- Setting up reporting hierarchy etc.

Managing Project

- Defining and setting up project scope
- Managing project management activities
- Monitoring progress and performance
- Risk analysis at every phase
- Take necessary step to avoid or come out of problems
- Act as project spokesperson

Software Management Activities

- Software project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management. Project management activities may include:
- **Project Planning**
- **Scope Management**
- **Project Estimation**

Project Planning

- Software project planning is task, which is performed before the production of software actually starts.
- It is there for the software production but involves no concrete activity that has any direction connection with software production; rather it is a set of multiple processes, which facilitates software production. Project planning may include the following:

Scope Management

- It defines the scope of project; this includes all the activities, process need to be done in order to make a deliverable software product. Scope management is essential because it creates boundaries of the project by clearly defining what would be done in the project and what would not be done. This makes project to contain limited and quantifiable tasks, which can easily be documented and in turn avoids cost and time overrun.
- During Project Scope management, it is necessary to -
- Define the scope
- Decide its verification and control
- Divide the project into various smaller parts for ease of management.
- Verify the scope
- Control the scope by incorporating changes to the scope

Project Estimation

- For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.
- Project estimation may involve the following:

Software size estimation

- Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

Effort estimation

- The Managers estimate efforts in terms of personnel requirement and man-hour required to produce the software.
- For effort estimation software size should be known.
- This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.

Time estimation

- Once size and efforts are estimated, the time required to produce the software can be estimated.
- Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS).
- The tasks are scheduled on day-to-day basis or in calendar months.
- The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

Cost estimation

- This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -
  - Size of software
  - Software quality
  - Hardware
  - Additional software or tools, licenses etc.
  - Skilled personnel with task-specific skills
  - Travel involved
  - Communication
  - Training and support

Project Configuration Management

- Configuration management is about to control the changes in software like requirements, design, and development of the product.
- The Primary goal is to increase productivity with fewer errors.

Some reasons show the need for configuration management:

- Several people work on software that is continually update.
- Help to build coordination among suppliers.
- Changes in requirement, budget, schedule need to accommodate.
- Software should run on multiple systems.

Tasks perform in Configuration management

- Identification
- Baseline
- Change Control
- Configuration Status Accounting
- Configuration Audits and Reviews

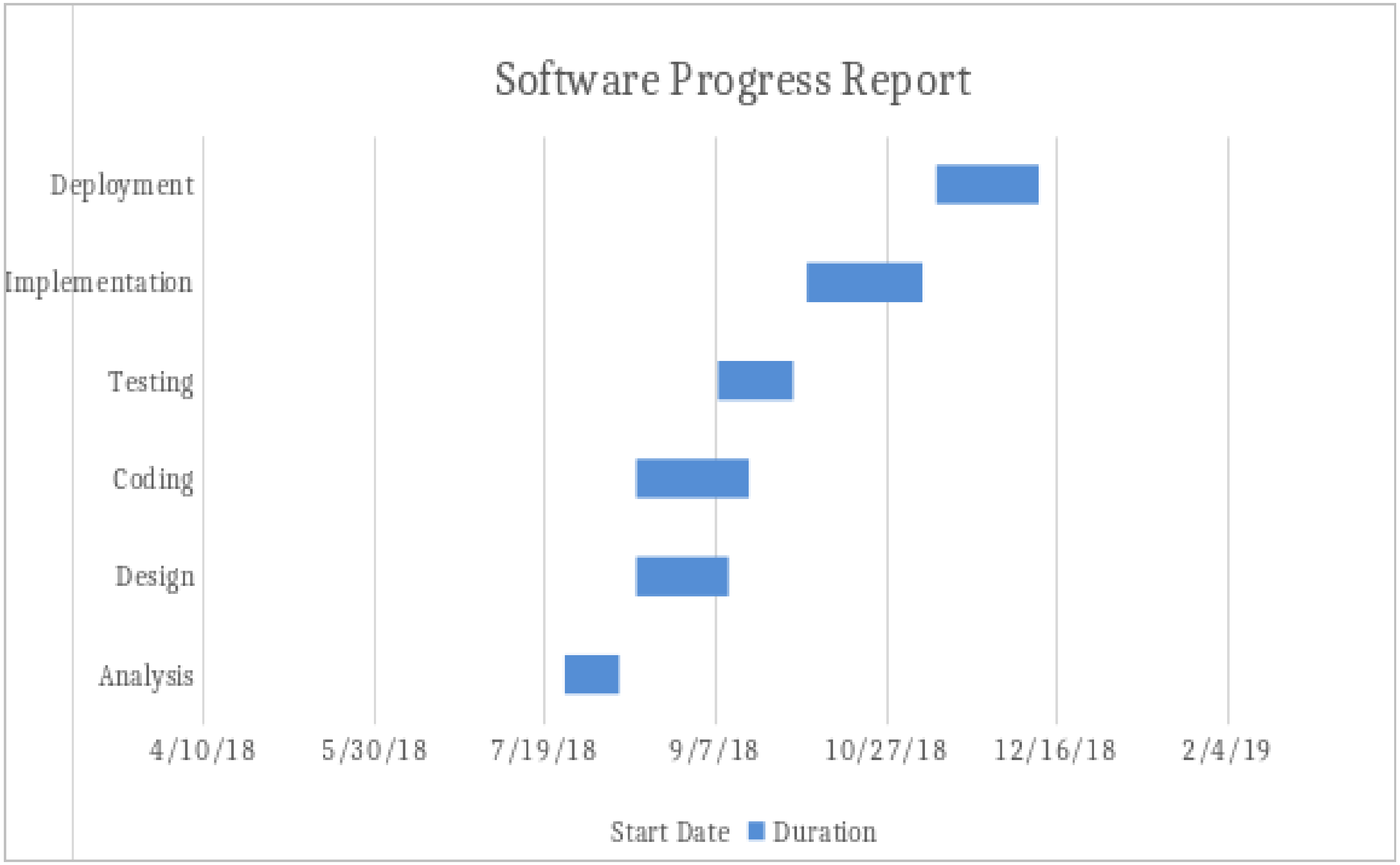
Project Management Tools

- Gantt chart
- PERT chart

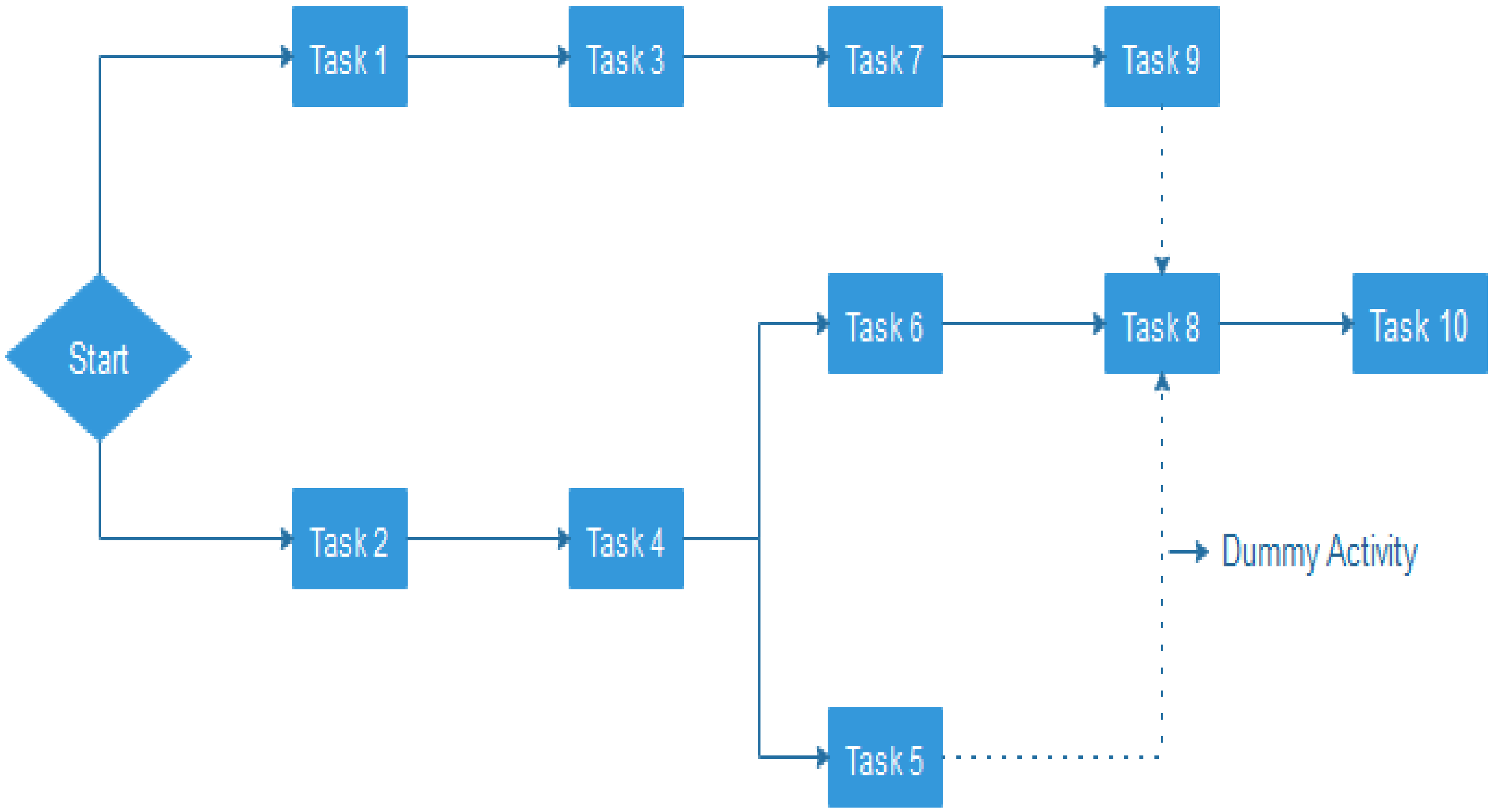
Gantt chart

- Gantt Chart first developed by Henry Gantt in 1917.
- Gantt chart usually utilized in project management, and it is one of the most popular and helpful ways of showing activities displayed against time.
- Each activity represented by a bar.
- Gantt chart is a useful tool when you want to see the entire landscape of either one or multiple projects.
- It helps you to view which tasks are dependent on one another and which event is coming up.

Gantt chart


Software Progress Report
Deployment
Implementation
Testing
Coding
Design
Analysis
4/10/18
5/30/18
7/19/18
9/7/18
10/27/18
12/16/18
2/4/19
Start Date
Duration

PERT chart


Start
Task 1
Task 3
Task 7
Task 9
Task 2
Task 4
Task 6
Task 8
Task 10
Task 5
Dummy Activity

PERT chart

- PERT is an acronym of Programme Evaluation Review Technique. In the 1950s, it is developed by the U.S. Navy to handle the Polaris submarine missile programme.
- In Project Management, PERT chart represented as a network diagram concerning the number of nodes, which represents events.
- The direction of the lines indicates the sequence of the task.
- In the above example, tasks between "Task 1 to Task 9" must complete, and these are known as a dependent or serial task.
- Between Task 4 and 5, and Task 4 and 6, nodes are not depended and can undertake simultaneously.
- These are known as Parallel or concurrent tasks. Without resource or completion time, the task must complete in the sequence which is considered as event dependency, and these are known as Dummy activity and represented by dotted lines.

Real-time Specialization

Objectives

- To explain the concept of a real-time system and why these systems are usually implemented as concurrent processes
- To describe a design process for real-time systems
- To introduce generic process architectures for monitoring and control and data acquisition systems

Topics covered

- System design
- Real-time operating systems
- Monitoring and control systems
- Data acquisition systems

Real-time systems

- Systems which monitor and control their environment.
- Inevitably associated with hardware devices
  - Sensors: Collect data from the system environment;
  - Actuators: Change (in some way) the system's environment;
- Time is critical. Real-time systems MUST respond within specified times.

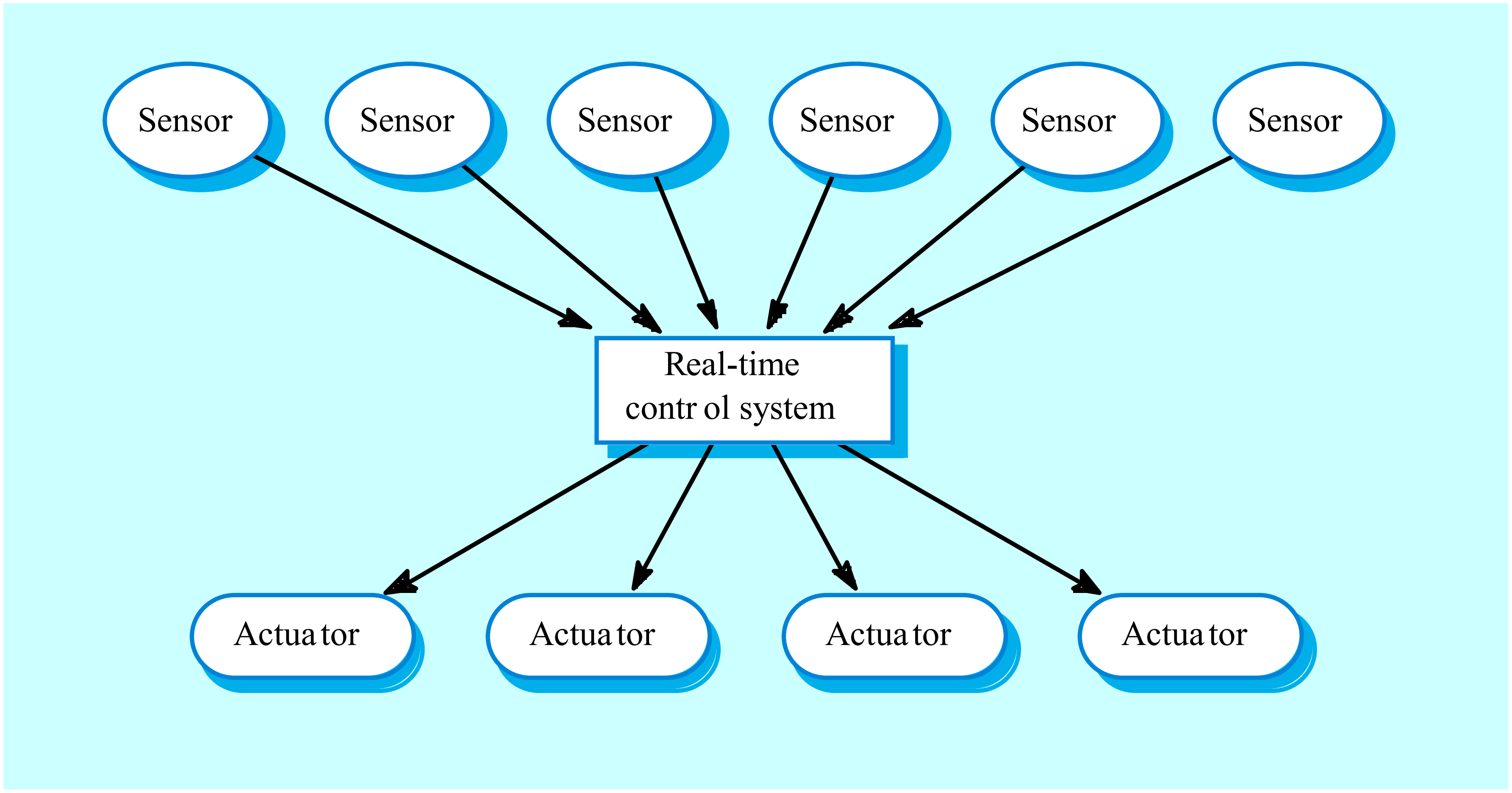
Definition

- A real-time system is a software system where the correct functioning of the system depends on the results produced by the system and the time at which these results are produced.
- A soft real-time system is a system whose operation is degraded if results are not produced according to the specified timing requirements.
- A hard real-time system is a system whose operation is incorrect if results are not produced according to the timing specification.

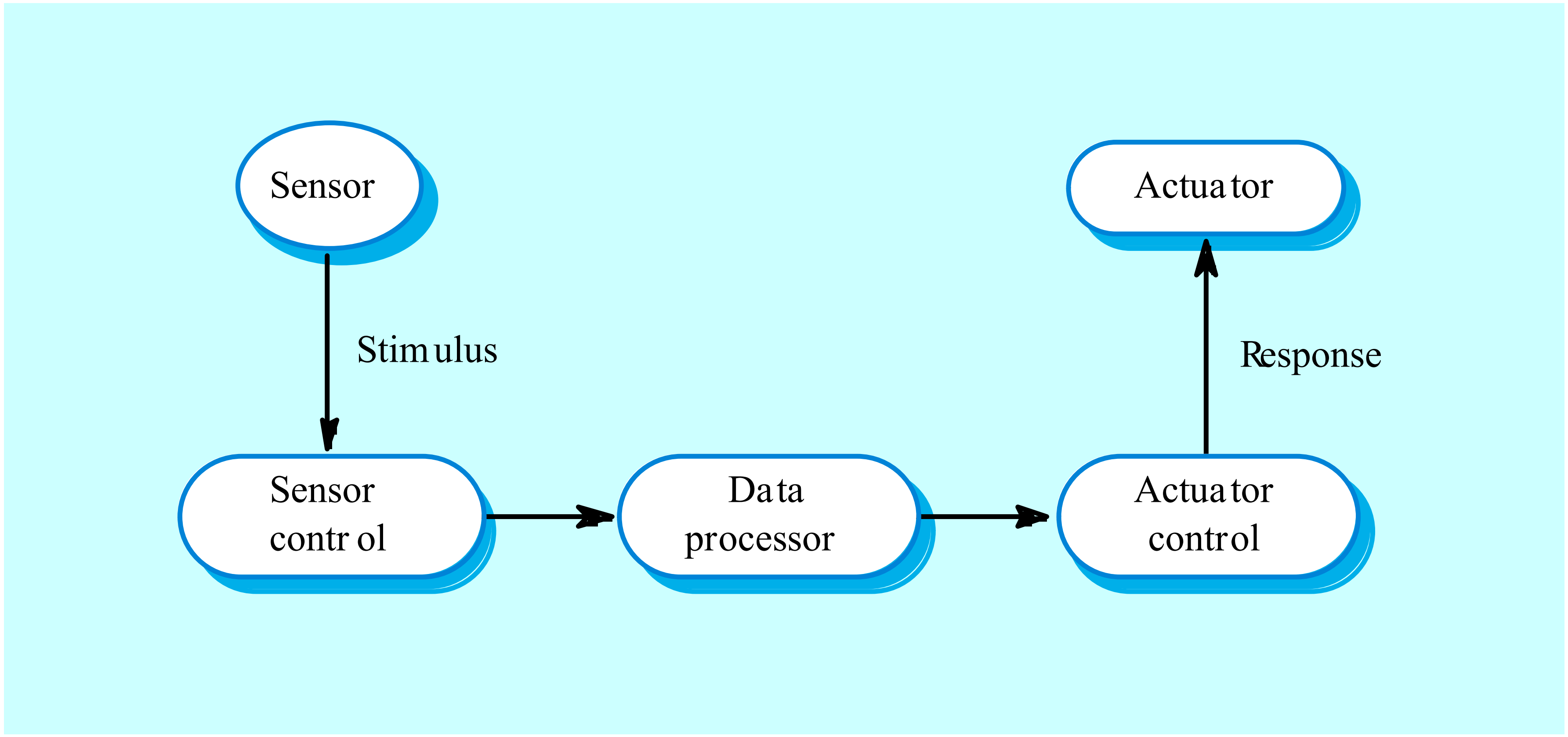
Architectural considerations

- Because of the need to respond to timing demands made by different stimuli/responses, the system architecture must allow for fast switching between stimulus handlers.
- Timing demands of different stimuli are different so a simple sequential loop is not usually adequate.
- Real-time systems are therefore usually designed as cooperating processes with a real-time executive controlling these processes.

A real-time system model



Sensor/actuator processes



System elements

- Sensor control processes
  - Collect information from sensors. May buffer information collected in response to a sensor stimulus.
- Data processor
  - Carries out processing of collected information and computes the system response.
- Actuator control processes
  - Generates control signals for the actuators.

Real-time programming

- Hard-real time systems may have to programmed in assembly language to ensure that deadlines are met.
- Languages such as C allow efficient programs to be written but do not have constructs to support concurrency or shared resource management.

Java as a real-time language

- Java supports lightweight concurrency (threads and synchronized methods) and can be used for some soft real-time systems.
- Java 2.0 is not suitable for hard RT programming but real-time versions of Java are now available that address problems such as
  - Not possible to specify thread execution time;
  - Different timing in different virtual machines;
  - Uncontrollable garbage collection;
  - Not possible to discover queue sizes for shared resources;
  - Not possible to access system hardware;
  - Not possible to do space or timing analysis.

System design

- Design both the hardware and the software associated with system. Partition functions to either hardware or software.
- Design decisions should be made on the basis on non-functional system requirements.
- Hardware delivers better performance but potentially longer development and less scope for change.

R-T systems design process

- Identify the stimuli to be processed and the required responses to these stimuli.
- For each stimulus and response, identify the timing constraints.
- Aggregate the stimulus and response processing into concurrent processes. A process may be associated with each class of stimulus and response.

R-T systems design process

- Design algorithms to process each class of stimulus and response. These must meet the given timing requirements.
- Design a scheduling system which will ensure that processes are started in time to meet their deadlines.
- Integrate using a real-time operating system.

Timing constraints

- May require extensive simulation and experiment to ensure that these are met by the system.
- May mean that certain design strategies such as object-oriented design cannot be used because of the additional overhead involved.
- May mean that low-level programming language features have to be used for performance reasons.

Real-time system modelling

- The effect of a stimulus in a real-time system may trigger a transition from one state to another.
- Finite state machines can be used for modelling real-time systems.
- However, FSM models lack structure. Even simple systems can have a complex model.
- The UML includes notations for defining state machine models

Monitoring and control systems

- Important class of real-time systems.
- Continuously check sensors and take actions depending on sensor values.
- Monitoring systems examine sensors and report their results.
- Control systems take sensor values and control hardware actuators.

Burglar alarm system

- A system is required to monitor sensors on doors and windows to detect the presence of intruders in a building.
- When a sensor indicates a break-in, the system switches on lights around the area and calls police automatically.
- The system should include provision for operation without a mains power supply.

Burglar alarm system

- Sensors
  - Movement detectors, window sensors, door sensors;
  - 50 window sensors, 30 door sensors and 200 movement detectors;
  - Voltage drop sensor.
- Actions
  - When an intruder is detected, police are called automatically;
  - Lights are switched on in rooms with active sensors;
  - An audible alarm is switched on;
  - The system switches automatically to backup power when a voltage drop is detected.

The R-T system design process

- Identify stimuli and associated responses.
- Define the timing constraints associated with each stimulus and response.
- Allocate system functions to concurrent processes.
- Design algorithms for stimulus processing and response generation.
- Design a scheduling system which ensures that processes will always be scheduled to meet their deadlines.

Key points

- Real-time system correctness depends not just on what the system does but also on how fast it reacts.
- A general RT system model involves associating processes with sensors and actuators.
- Real-time systems architectures are usually designed as a number of concurrent processes.

Key points

- Monitoring and control systems poll sensors and send control signal to actuators.
- Data acquisition systems are usually organised according to a producer consumer model.

Summary

- The development of advanced industrial real-time systems is one of the major areas of applicability for OOSE.
- OOSE does not bind the solution space of real-time applications in the beginning of the development.
- That is, the problem can be analyzed, to a large extent, independently of the artifacts such as operating systems, languages etc., that will be used in its realization.
- A real-time system typically detects and/or controls events outside the system under timing constraints.

Summary

- The main point is that real-time systems adds an additional dimension to the system, namely time. This makes these types of system harder to develop.
- The fundamental issues of real-time systems includes processes, synchronization, and communication. OOSE needs to be specialized to handle real-time systems, since it does not include any modeling techniques that handle this specifically.
- This means one should use the semantics of the actual implementation environment.

Summary

- Analysis is not affected by real-time issues. However, during construction the system needs to be designed and implemented under these constraints.
- This often involves the design of processes, communication and synchronization.
- Here the use cases provide a strong tool since they express the execution, or use, of the system.

Object Oriented Analysis(OOA)

Click to add text By Coad and Yourdon .

Presented By:
Rashmi Neupane [21]
Apil Adhikari[22]

Table of Content:

Introduction

- The main idea in this method is to extend the model with respect to processes, human interfaces and DBMS issues.
- Coad Yourdon purpose a model or architecture that is partitioned horizontally into layer and vertically into components.
- P. Coad and E .Yourdon purposed this model.
- It has primary strength in system analysis.
- Generally the model is based on
  - ER Diagram
  - Object Oriented Programming and Knowledge Base System
  - Information Modeling

Major Activities :

- Finding Classes and Object
- Identifying Structures
- Identifying Subjects
- Defining Attributes and Instance Connection
- Defining Services

Contd.

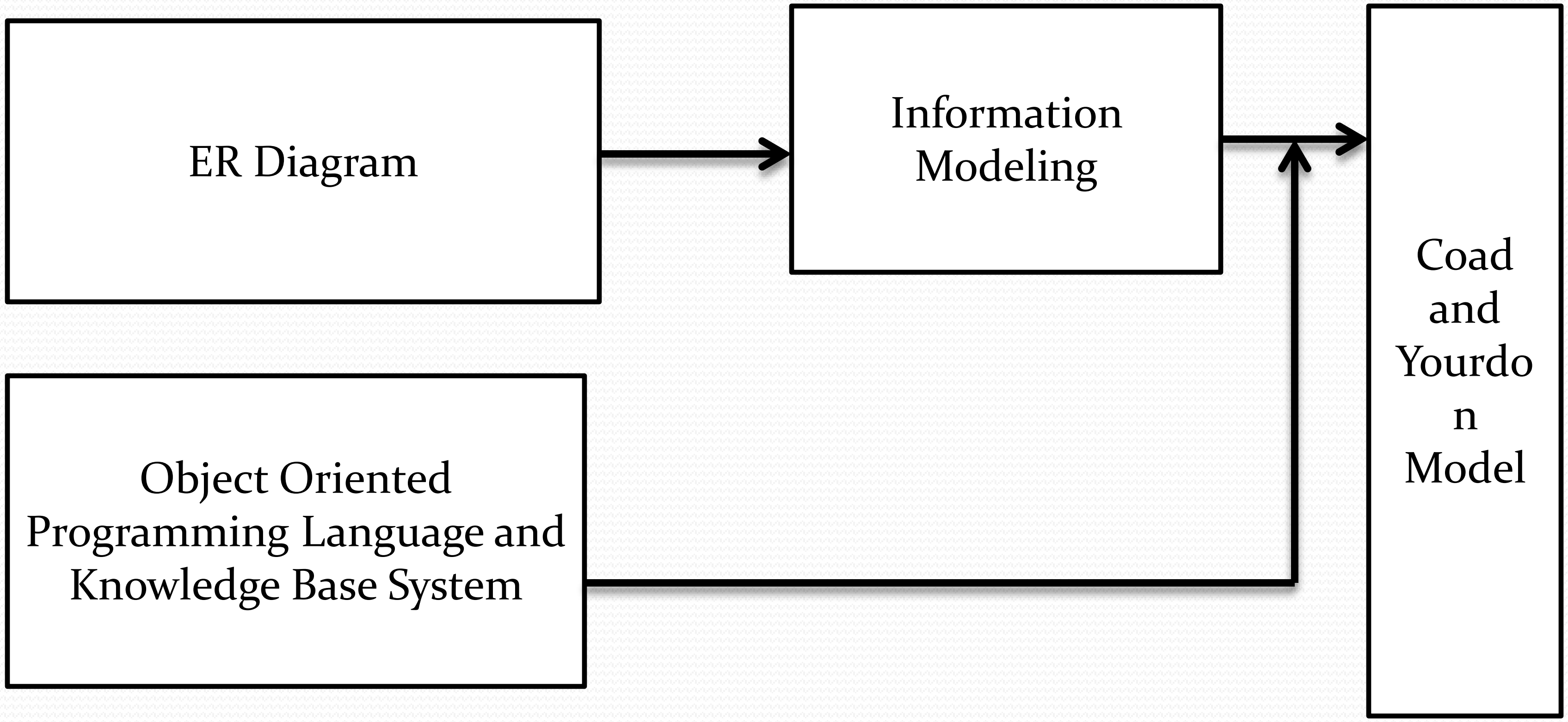

ER Diagram
Information Modeling
Object Oriented Programming Language and Knowledge Base System
Coad and Yourdon Model

Fig. Coad and Yourdon Model

Contd.

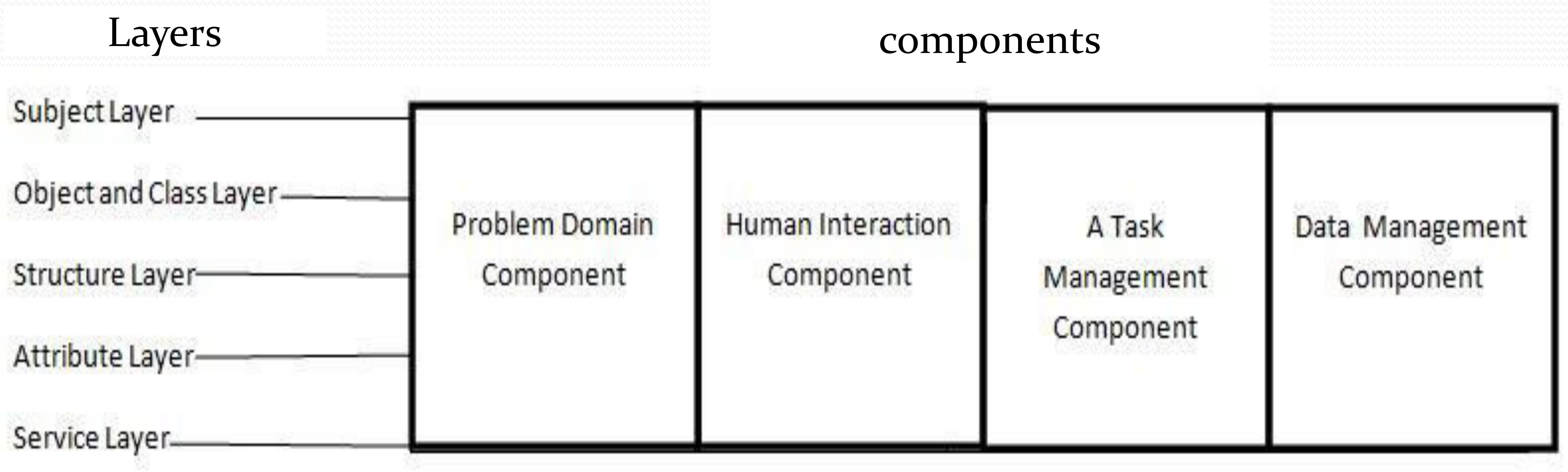


Fig : Architecture/Methodology of Coad Yourdon Analysis Model

(Partitioned horizontally into layers and vertically into components)

Analysis Phase of Coad Yourdon Model

- This methodology is known as SOSAS.
- **S→ Subject Layer:**

  defined the subject included in the system.
- **O→ Object and Class Layer:**

  identify the object classes and the class hierarchies.
- **S→ Structure Layer:**

  displays the system generalization/specifications and whole -part structure.

  two types of structures are classification and composition structure. classification instances handle the instance connection between related classes , while composition structure handle all of other connection over classes.
- **A→ Attribute Layer:**

  displays instance connection (association relation between classes)
- **S→ Service Layer:**

  all the behavior ,methods and message connection for each class are identified.

Design Phase Of Coad Yourdon Model

- **Problem Domain Component**

  The PDC includes the problem domain classes that were identified during object oriented analysis.

- **Human Interaction Component**

  includes interface classes between or among Objects.

- **A Task Management Component**

  a task can be defined as process or stream of activity .(system wide management activities are included)

- **Data Management Component**

  Defines a classes needed for database access methods.

Design Evaluation Criteria

- For correct and effective design.
- Evaluation Criteria:

➢ **Coupling**

Describes the level of interaction between or among components . Two types of coupling

-**Interaction Coupling:** refers to connection between classes due to message connections.

-**Inheritance Coupling :** refers to the connection between generalization and specifications in generalization-specifications structures.

Contd.

- **Cohesion**

  It is a degree to which the element of a portion of a design contribute to the carrying out a single well defined purpose.

Three types of cohesion:

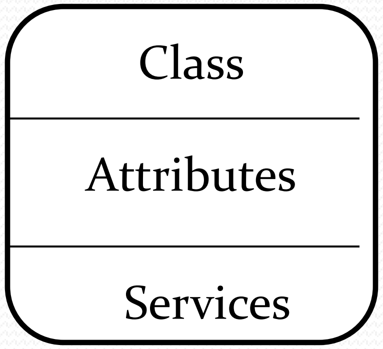
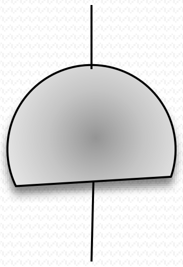
1. Service
2. Class
3. Generalization/Specification Cohesion

- **Reuse**

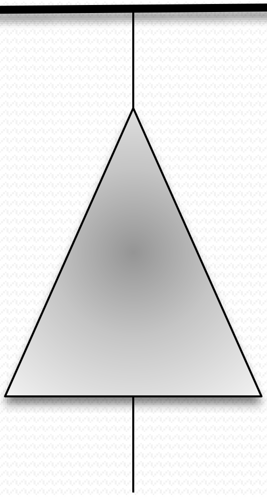
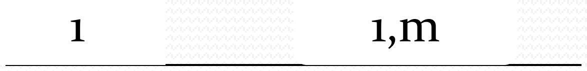
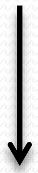
  During object oriented development design results, analysis result etc can be reused.

Coad Yourdon Notation

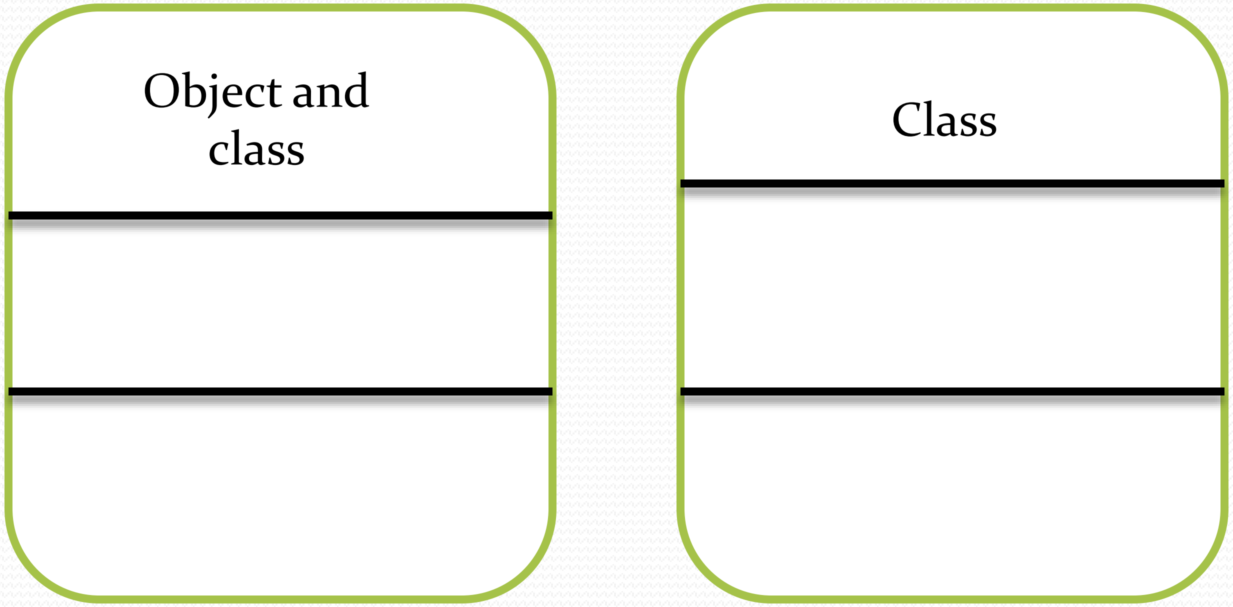
Notations:

Symbol	Meanings
	OOA Class Symbol
	Generalization/inheritance

Contd.

	Whole Part -Structure Aggregations
	Instance Connectivity (one to many)
	Transition

Contd.



Class and object Notation and Class Notation

Contd.

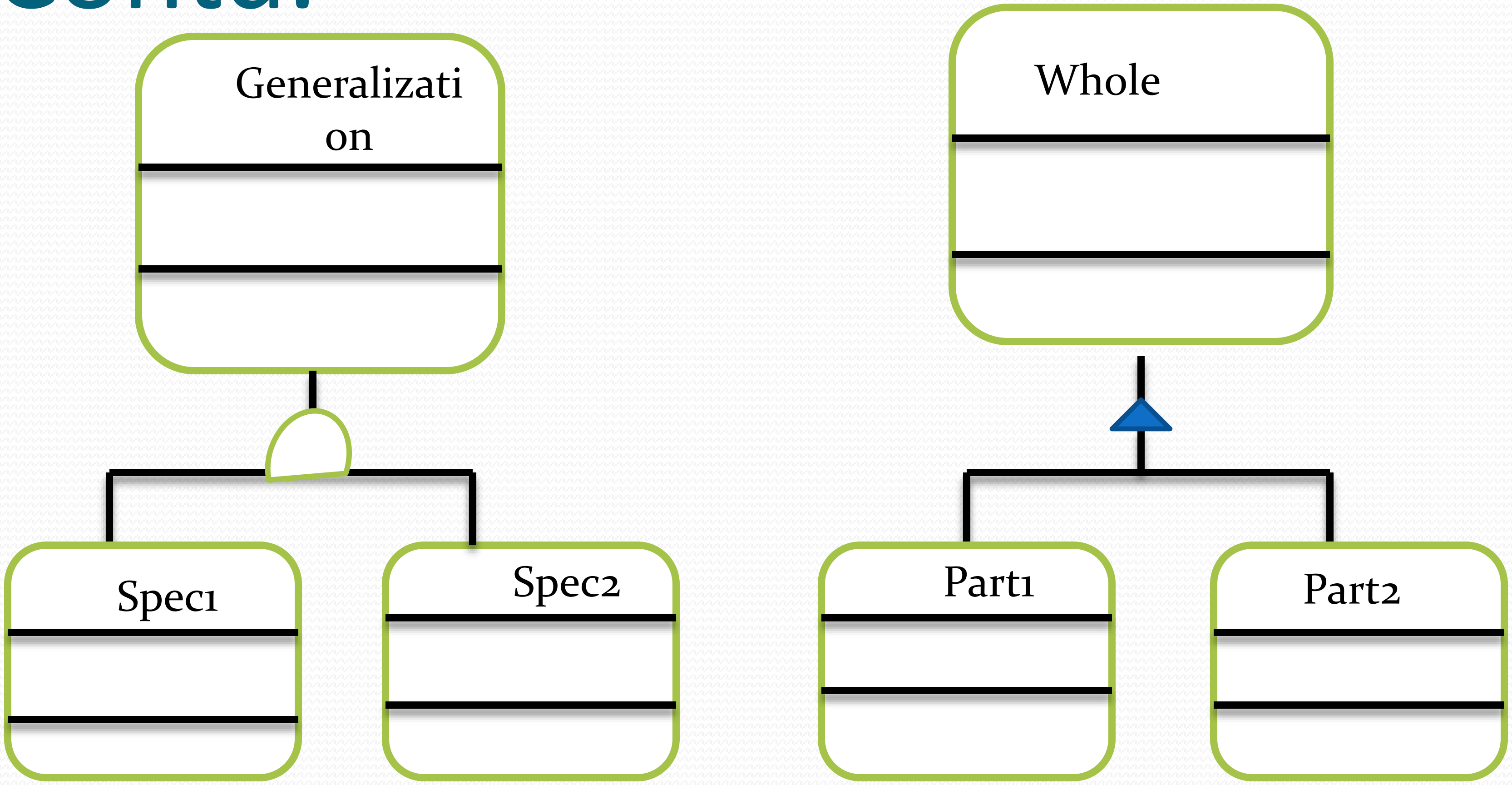
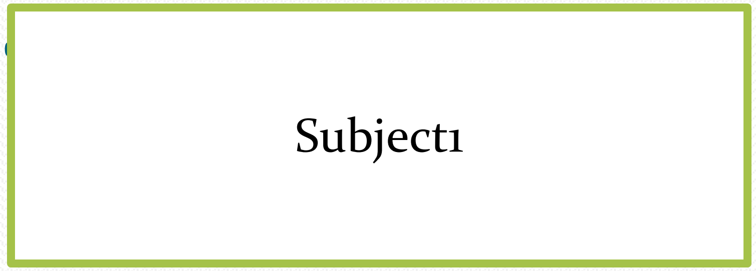
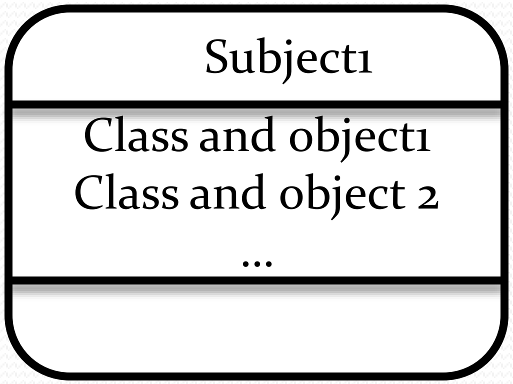
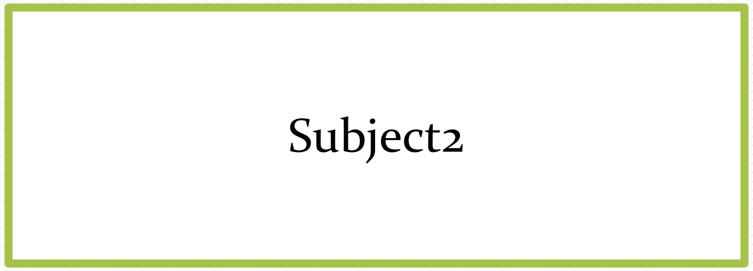
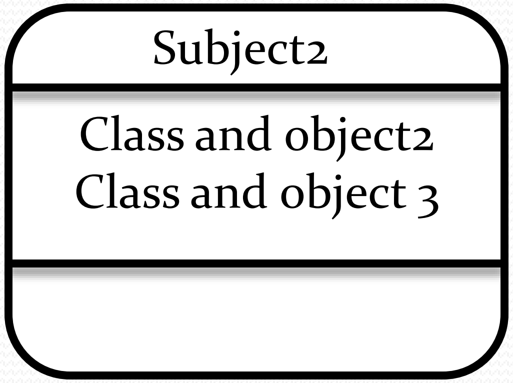


Fig . Gen/Specialization and Whole part Structure





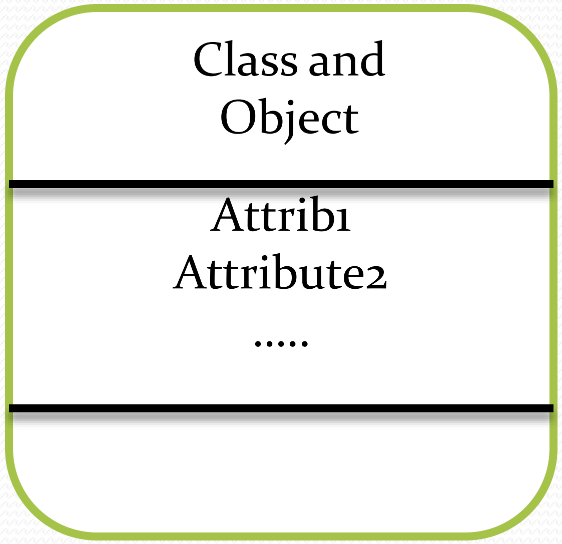




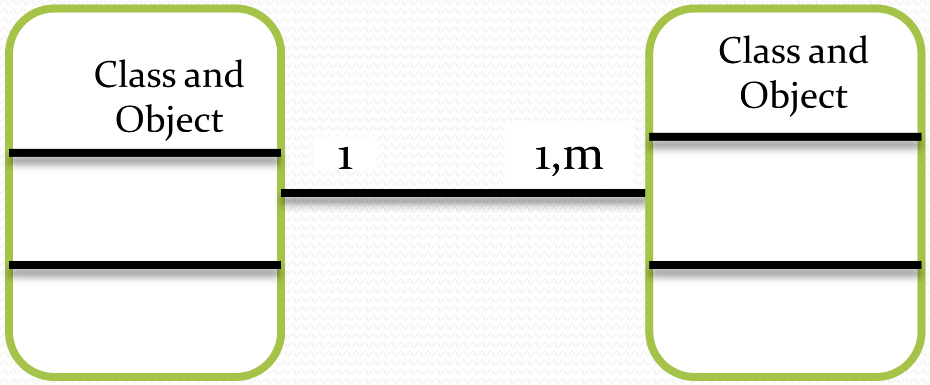
Subject Notation before expand

Subject Notation after partially expand

Contd.

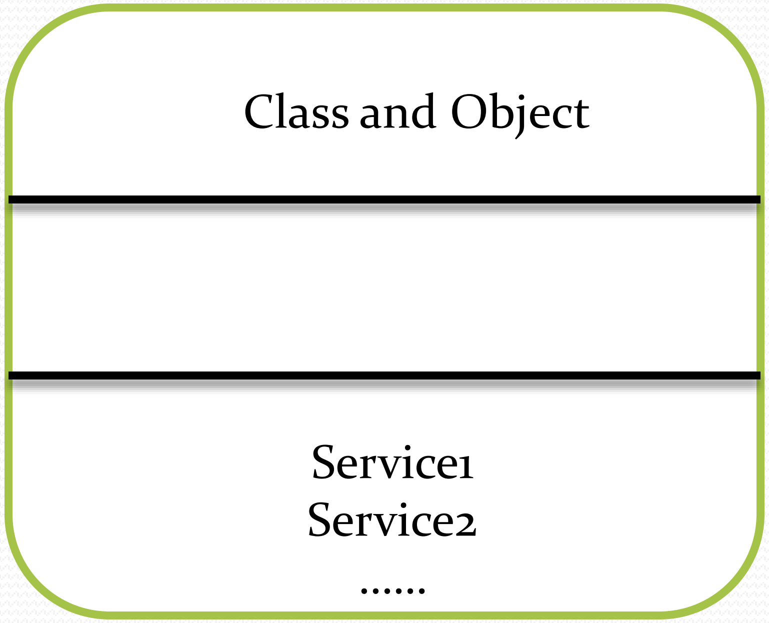


Defining Attributes



Instance Connection

Contd.



Defining Services

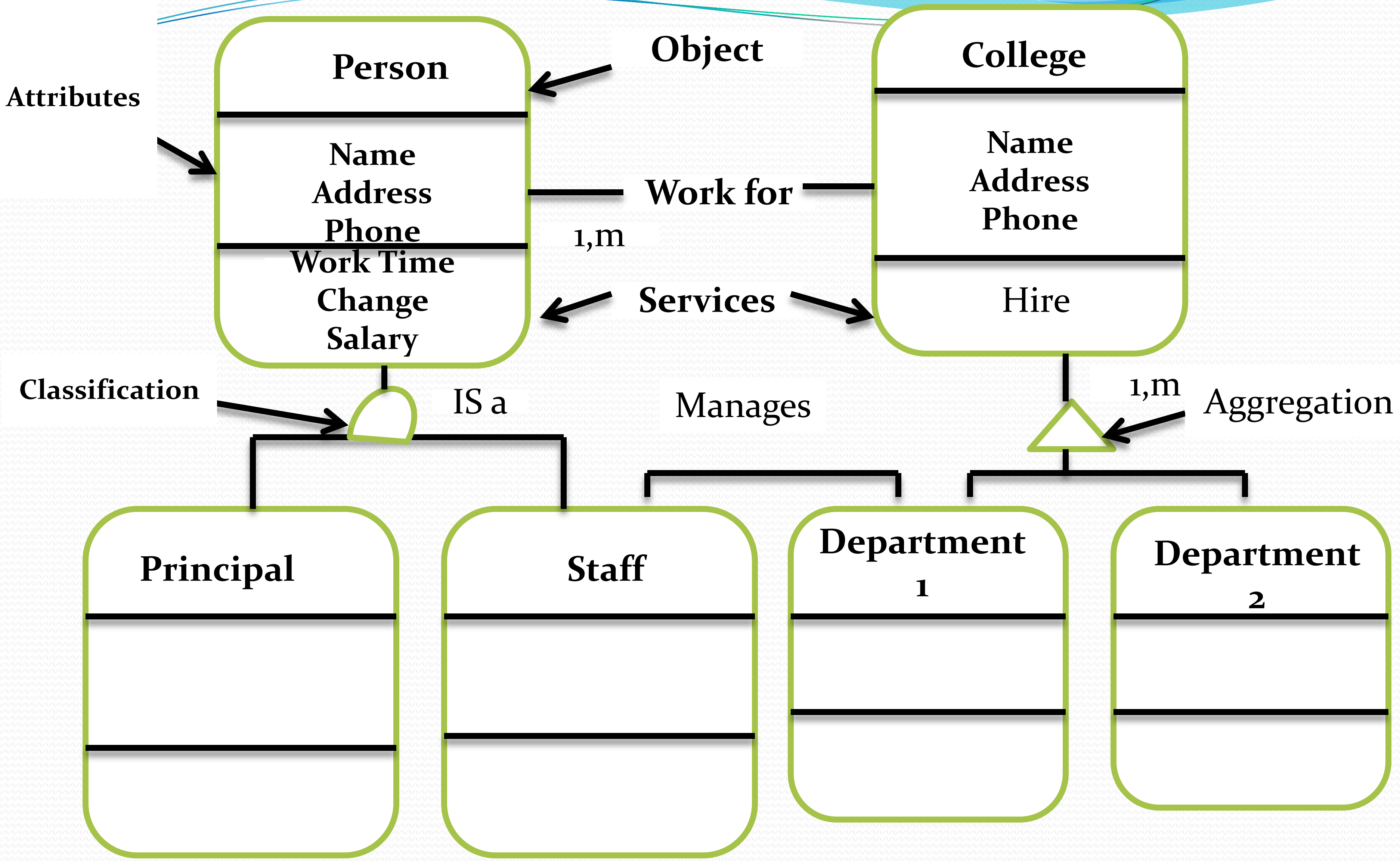
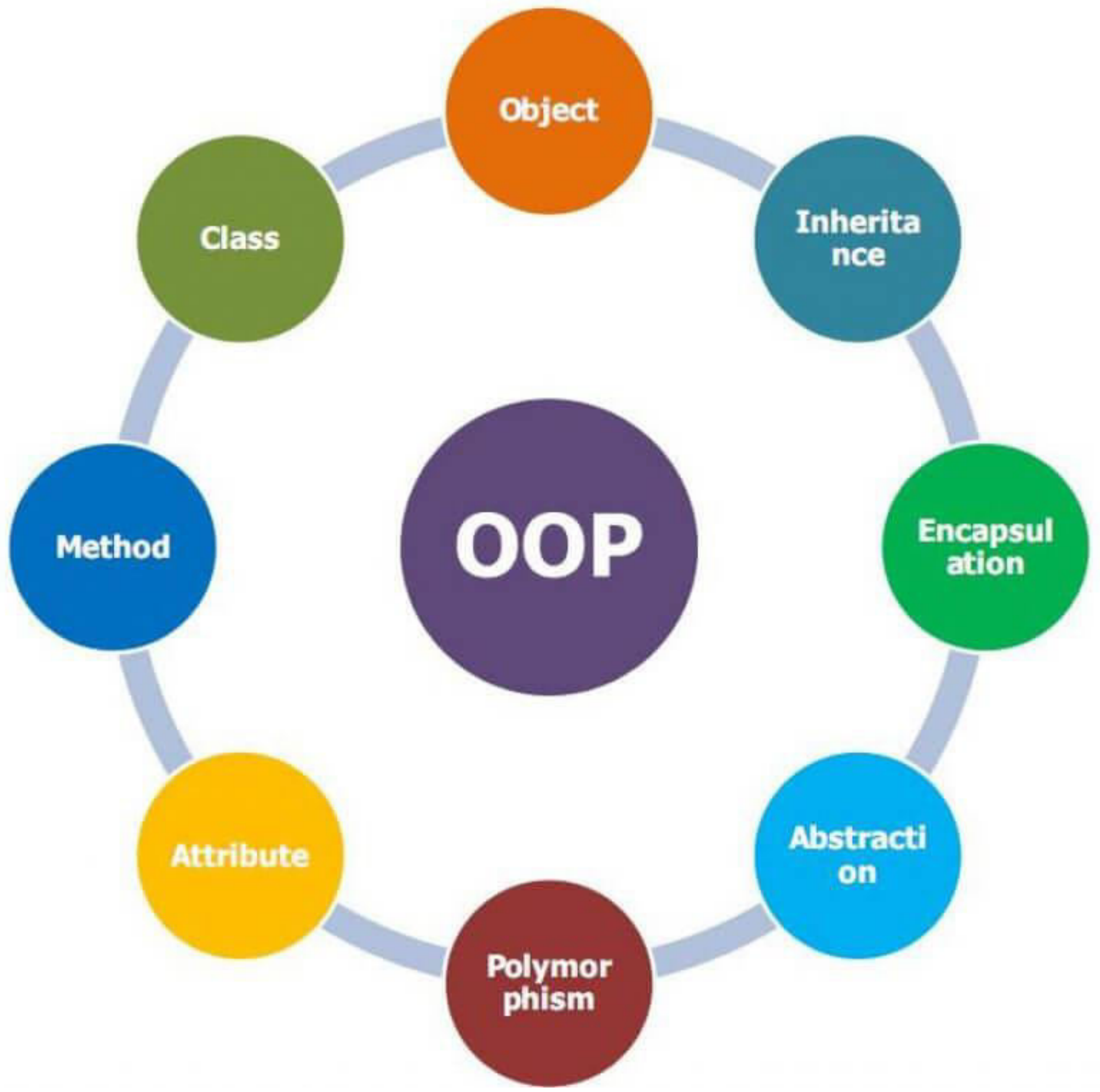


Fig. Case Study of College

Thank You

What is Object-Orientation?



By:

Rishav Acharaya

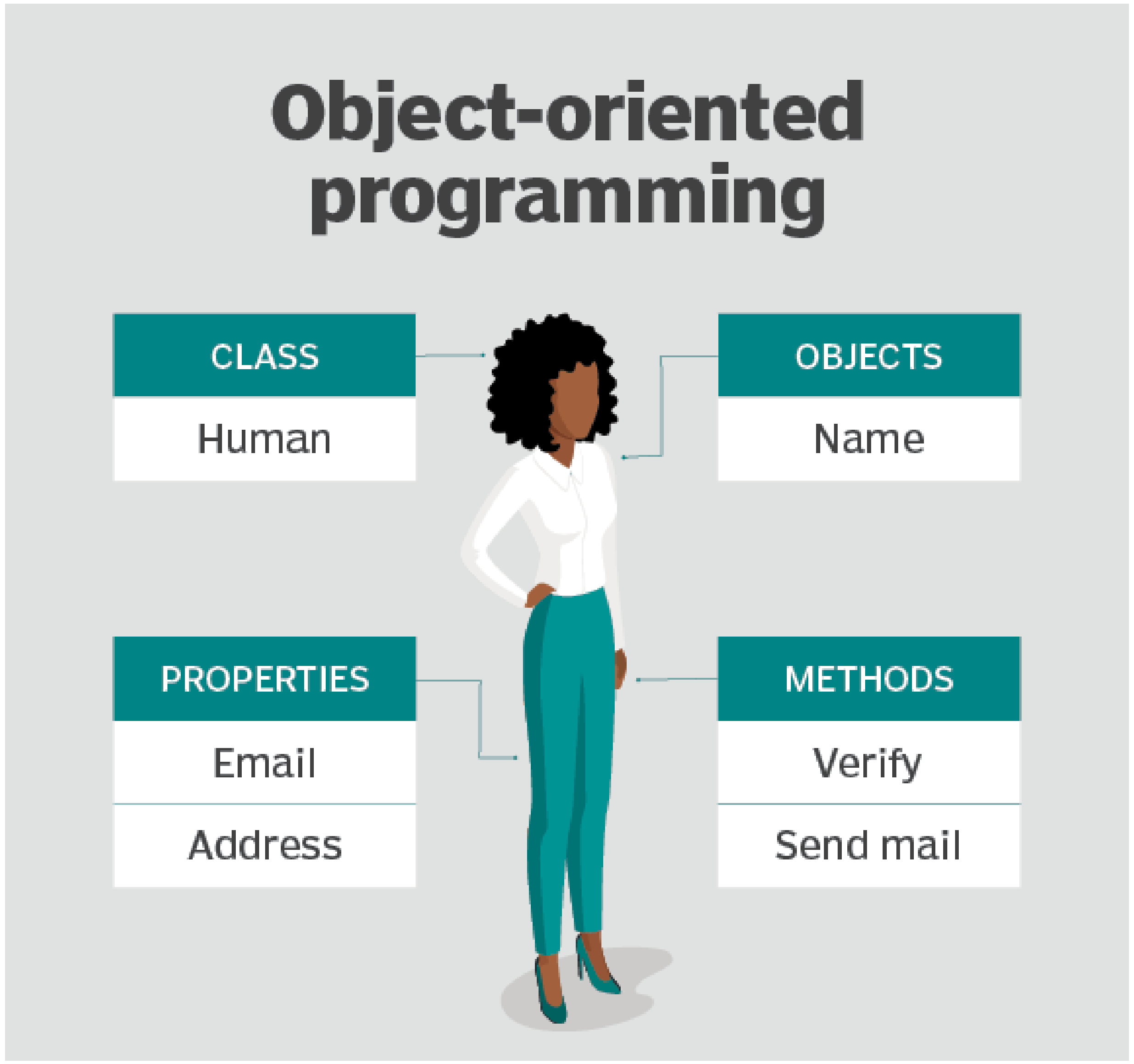
Sudhan Kandel

Table of content

Introduction

- ❖ Object orientation is a modeling and development methodology based on object-oriented (OO) concepts.
- ❖The object-oriented approach, however, focuses on objects that represent abstract or concrete things in the real world.

Structure of OOP



What are the benefits of OOP?

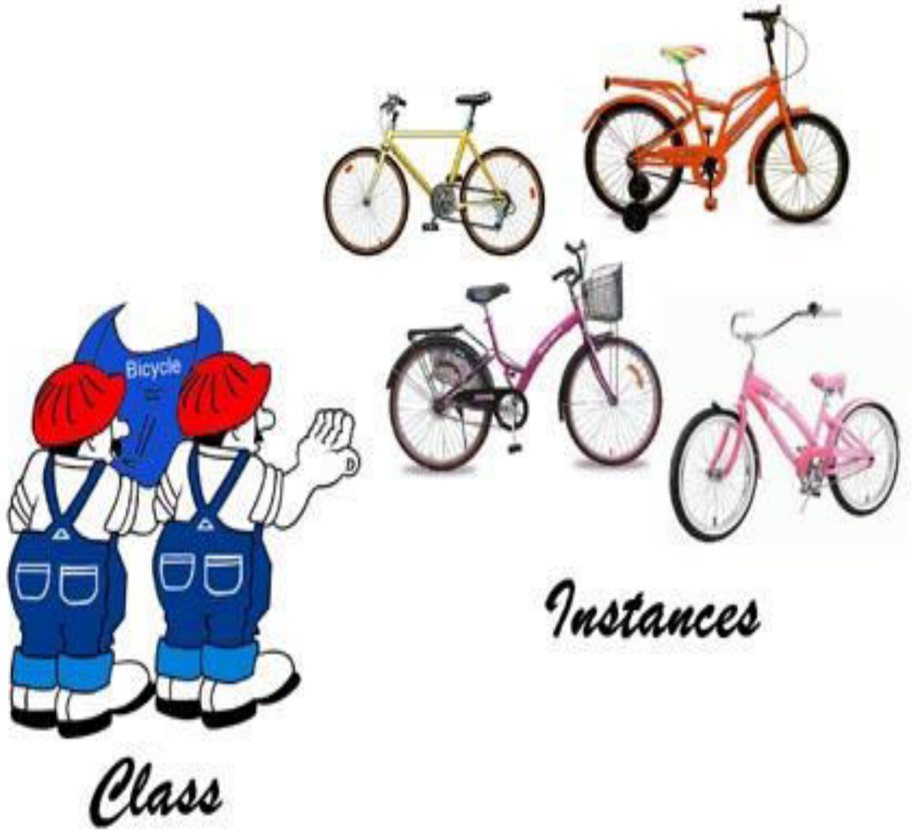
- Modularity
- Reusability
- Productivity
- Easily upgradable and scalable
- Security
- Flexibility

What is Object

- ❖Objects are instances of classes
- ❖They contain data and provides services. The data forms the attributes of the object. The services are known as methods.

Class and Instance

- ❖A class is a user-defined data-type which has data members and member functions.
- ❖Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behavior of the objects in a Class.
- ❖ class Car, the data member will be speed limit, mileage etc. and member functions can apply brakes, increase speed etc.



Polymorphism

- Polymorphism is that in which we can perform a task in multiple forms or ways.
- It is applied to the functions or methods.
- It allows the object to decide which form of the function to implement at compile-time as well as run-time
- Type of polymorphism
    1. Compile-time polymorphism(Method Overloading)
    2. Run-time polymorphism (Metthod Overriding)

Contn..

- **Overloading** occurs when two or more methods in one class have the same method name but different parameters.
- **Overriding** occurs when two methods have the same method name and parameters. One of the methods is in the parent class, and the other is in the child class. Overriding allows a child class to provide the specific implementation of a method that is *already* present in its parent class.

Inheritance

- Inheritance is one in which a new class is created that inherits the properties of the already exist class.
- It supports the concept of code reusability and reduce the length of the code.

Type of Inheritance

Single Inheritance

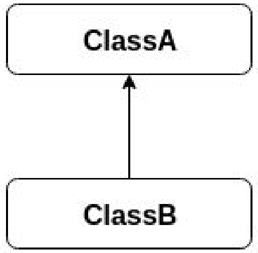
multi-level inheritance

Multiple inheritance

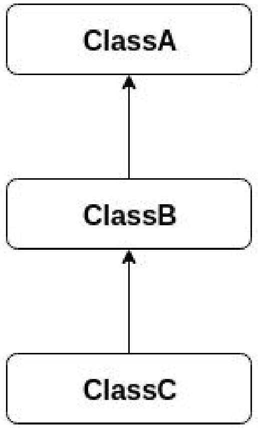
Hybrid inheritance

Hierarchical inheritance

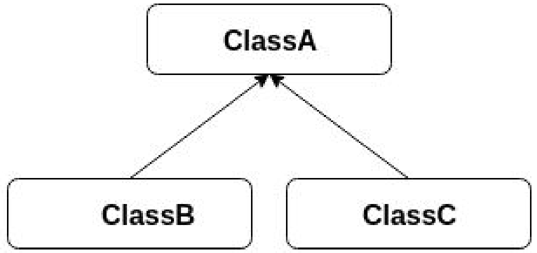
cont.



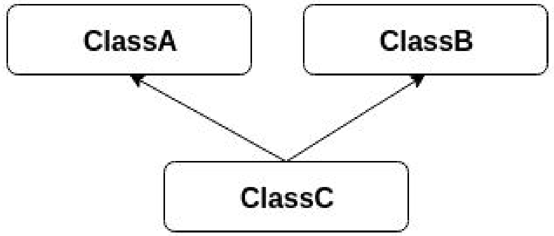
Single Inheritance



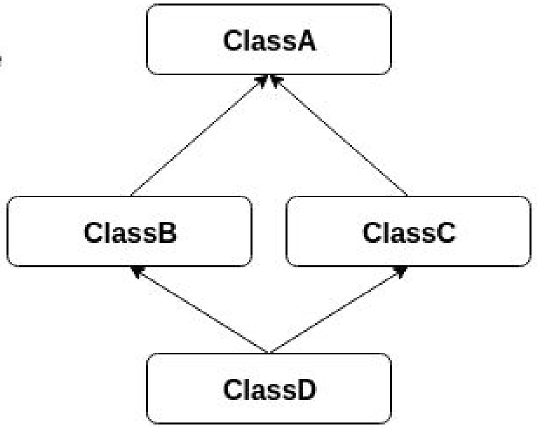
Multilevel Inheritance



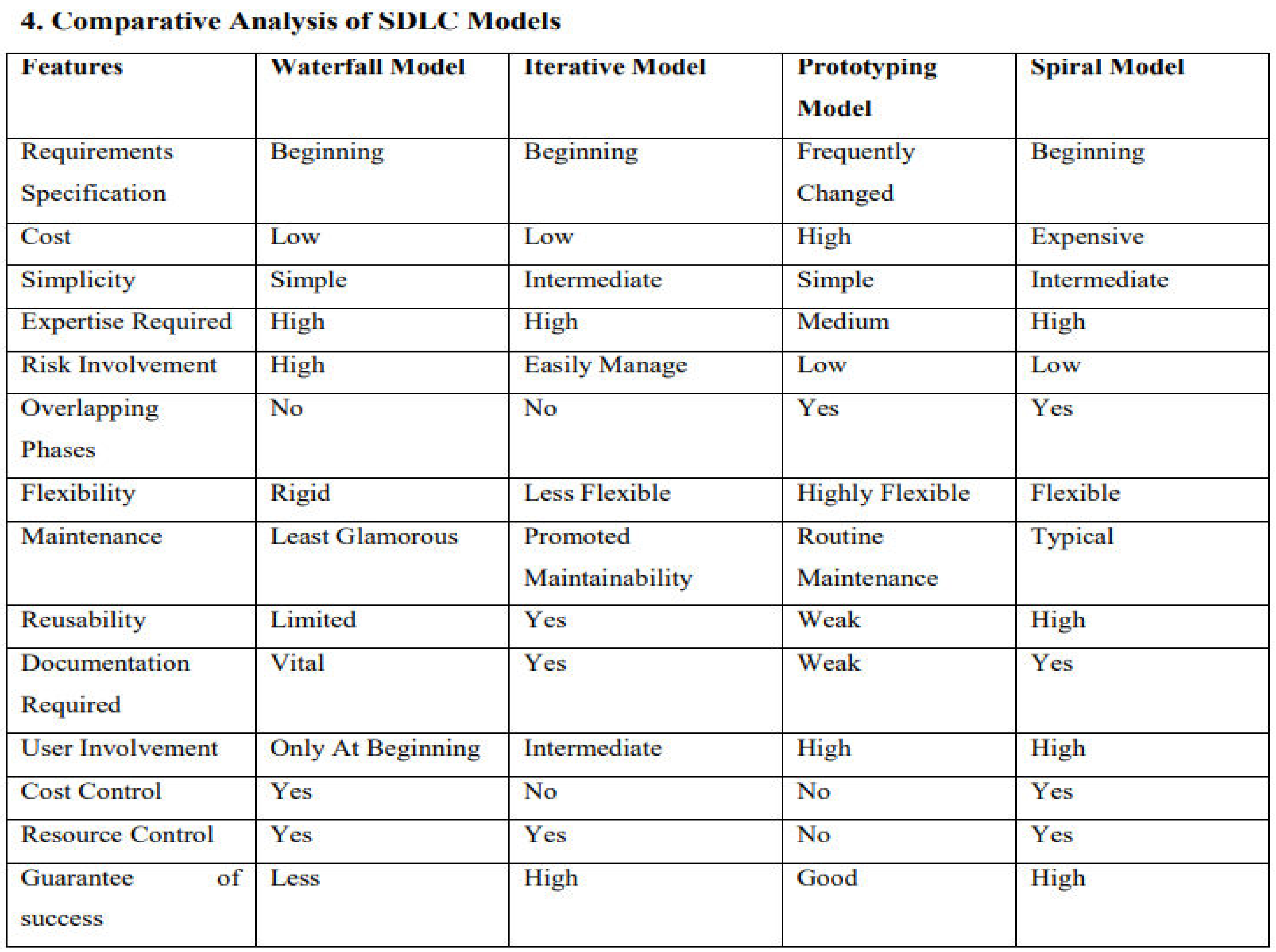
Hierarchical Inheritance



Multiple Inheritance



Hybrid Inheritance



4. Comparative Analysis of SDLC Models

Features	Waterfall Model	Iterative Model	Prototyping Model	Spiral Model
Requirements Specification	Beginning	Beginning	Frequently Changed	Beginning
Cost	Low	Low	High	Expensive
Simplicity	Simple	Intermediate	Simple	Intermediate
Expertise Required	High	High	Medium	High
Risk Involvement	High	Easily Manage	Low	Low
Overlapping Phases	No	No	Yes	Yes
Flexibility	Rigid	Less Flexible	Highly Flexible	Flexible
Maintenance	Least Glamorous	Promoted Maintainability	Routine Maintenance	Typical
Reusability	Limited	Yes	Weak	High
Documentation Required	Vital	Yes	Weak	Yes
User Involvement	Only At Beginning	Intermediate	High	High
Cost Control	Yes	No	No	Yes
Resource Control	Yes	Yes	No	Yes
Guarantee of success	Less	High	Good	High

Table1. Comparison of SDLC Models

Thank You

Responsibility Driven Design

- An object-oriented design methodology that focuses on assigning responsibilities to objects and defining the collaboration among objects.

Key Concept

- Responsibilities
  - Actions an object must perform
  - Knowledge an object maintains
  - Collaborations an object participates in
- Behavioral Focus
  - Emphasis on doing rather than having
  - Objects as active participants
  - Dynamic interaction patterns

Process/Design Step of Responsibility-Driven Design (RDD)

1. Identify Key Objects

- Think about real-world roles or concepts in the system
- Turn them into potential objects/classes

2. Assign Responsibilities

- Decide what each object should **know** and **do**
- Focus on behavior, not just data

Process/Design step of Responsibility-Driven Design (RDD)

3. Define Collaborations

- Determine how objects will **work together**
- Who sends messages to whom?

4. Use CRC Cards (optional)

- **C**lass
- **R**esponsibilities
- **C**ollaborators
- A simple tool to model roles and communication

Comparison between RDD and Traditional approach

Aspect	RDD	Traditional
Focus	Behavior and responsibility	Data structure and attributes.
Object Definition	What it does	What it knows
Design process	Bottom-up, iterative	Top-down, linear
Coupling	Loose, based on behavior	Tight, based on data
Testing	Behavior-driven	State-driven

Best Practices of RDD

- **Responsibility Assignment**
  - Keep responsibilities focused and single-minded.
  - Maintain clear boundaries between objects.
  - Avoid God objects with too many responsibilities.
- **Collaboration Design**
  - Minimize dependencies.
  - Use interfaces for flexibility.
  - Keep collaborations explicit.

Best Practices of RDD

- **Documentation**
  - Document responsibilities clearly
  - Show collaboration patterns
  - Maintain behavioral documentation

Benefit of RDD

- Improved maintainability.
- Enhanced Collaboration.
- Better Testing.

Common applications of RDD

- Domain Modeling.
- UI design.
- Service Architecture.