The Software Product

By
Mr. Priyajit Sen
Assistant Professor
Directorate of Distance Education
Vidyasagar University, Midnapore

- **Introduction**

- **Software**

Software is more than just a program code. A program is an executable code, which serves some computational purpose.

- **Characteristics of software**

  1) Software is developed or engineered; it is not manufactured in the classical sense:

a) Some similarities exist between software development and hardware manufacturing, but few activities are fundamentally different.

b) High quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems than software.

2) Software doesn’t —wear out:

a) Hardware components suffer from the growing effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out.

b) Software is not susceptible to the environmental maladies that cause hardware to wear out.

c) When a hardware component wears out, it is replaced by a spare part. There are no software spare parts.

3) Industry demands component-based construction, most software continues to be custom built:

a)A software component should be designed and implemented so that it can be reused in many different programs.

b) Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable parts.

- **Software myths**

31.3.1 Management myths:

Myth 1: We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Myth 2: If we get behind schedule, we can add more programmers and catch up.

Myth 3: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

- **Customer myths:**

Myth 1: A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.

Myth 2: Software requirements continually change, but change can be easily accommodated because software is flexible.

- **Practitioner’s myths :**

Myth 1: Once we write the program and get it to work, our job is done.

Myth 2: Until I get the program —running‖ I have no way of assessing its quality.

Solution 1: Technical Review

Solution 2: Software reviews are a —”quality filter” that have been found to be more effective than testing for finding certain classes of software defects.

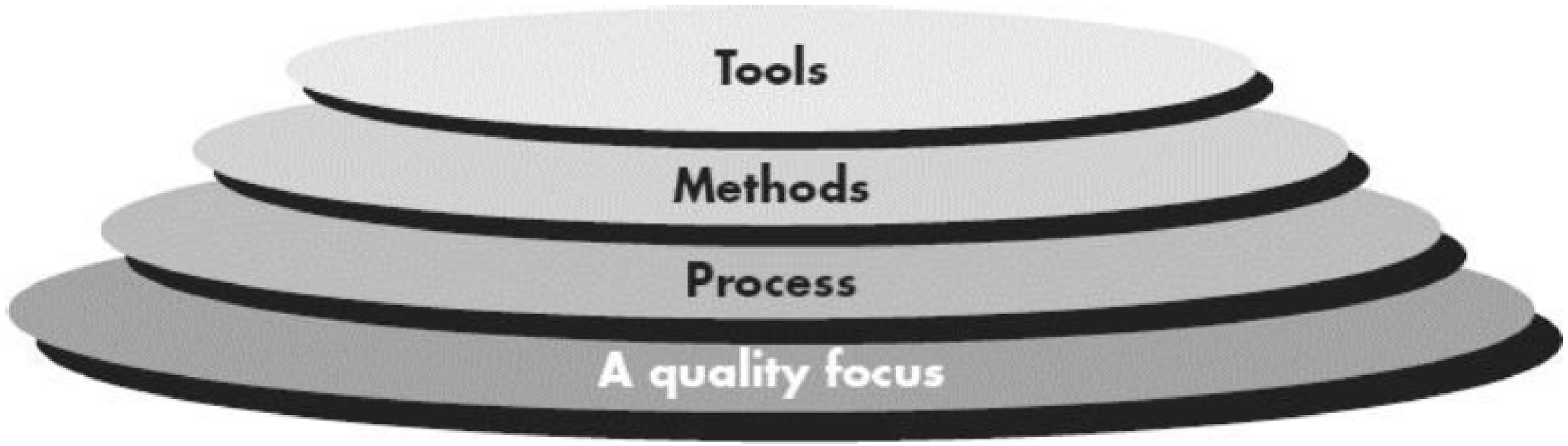
Myth 3: The only deliverable work product for a successful project is the working program.

- **Software Engineering**

  1) A systematic collection of good program development practices and techniques.

  2) Software engineering discusses systematic and cost-effective techniques for software development. These techniques help develop software using an engineering approach.

- **A Layered Technology:**

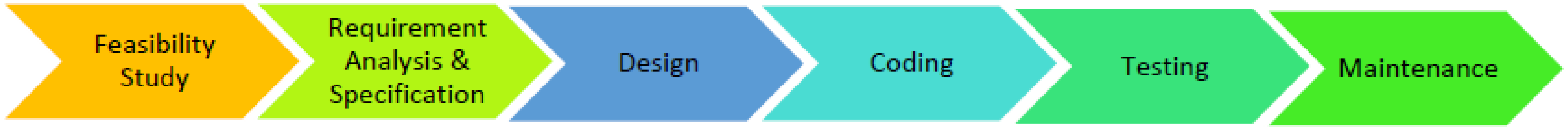


- **A Quality Focus:**
  - Every organization is rest on its commitment to quality.
  - Total quality management, Six Sigma, or similar continuous improvement culture and it is this culture ultimately leads to development of increasingly more effective approaches to software engineering.

- **Process:**
  - The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
  - The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products are produced and the quality is ensured.

- **Methods:**
  - Software engineering methods provide the technical aspects for building software.
  - Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.
- **Tools:**
  - Software engineering tools provide automated or semi-automated support for the process and the methods.
  - When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called CASE (computer-aided software engineering), is established.

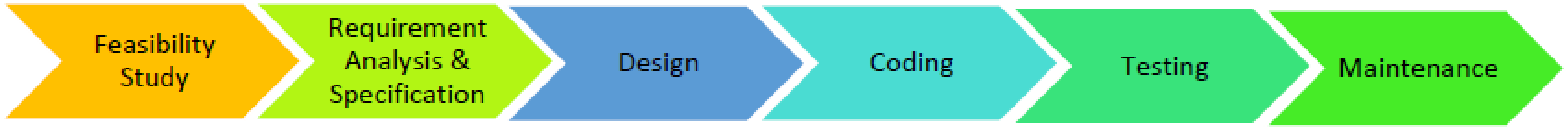
- **Software life cycle / Software development life cycle (SDLC)**



Phase1: Feasibility Study

1. **Technical Feasibility**
2. **Economic Feasibility**
3. **Operational Feasibility**
4. **Legal Feasibility**

- **Software life cycle / Software development life cycle (SDLC)**



Phase 2: Requirement Analysis and Specification.

1. Needs Analysis

2. Data Gathering

2.1 Written Documents

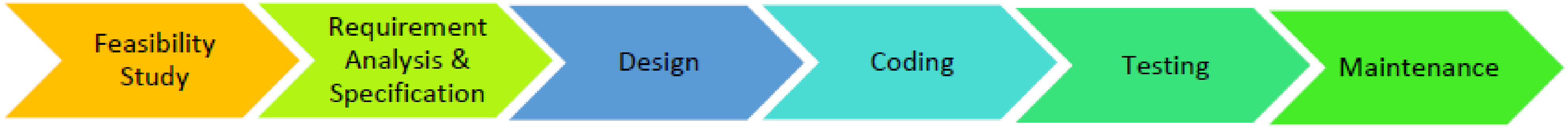
2.2 Interviews

2.2.1 Structured Interviews

2.2.2 Unstructured Interviews

2.3 Questionnaires

- **Software life cycle / Software development life cycle (SDLC)**



Phase 2: Requirement Analysis and Specification.

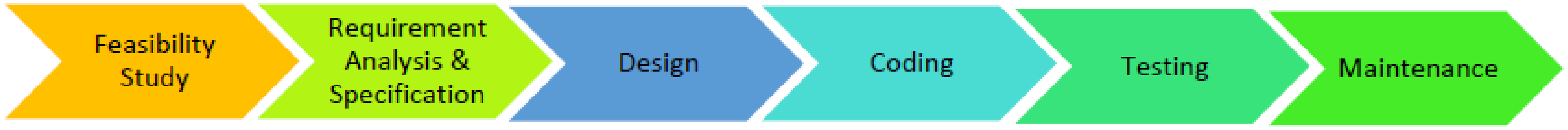
2.4 Observations

2.5 Sampling

3. Data Analysis

4. Analysis Report

- **Software life cycle / Software development life cycle (SDLC)**



Phase 3: Design

1. High-level design

2. Low-level design

Phase 4: Coding

Phase 5: Testing

Phase 6: Maintenance

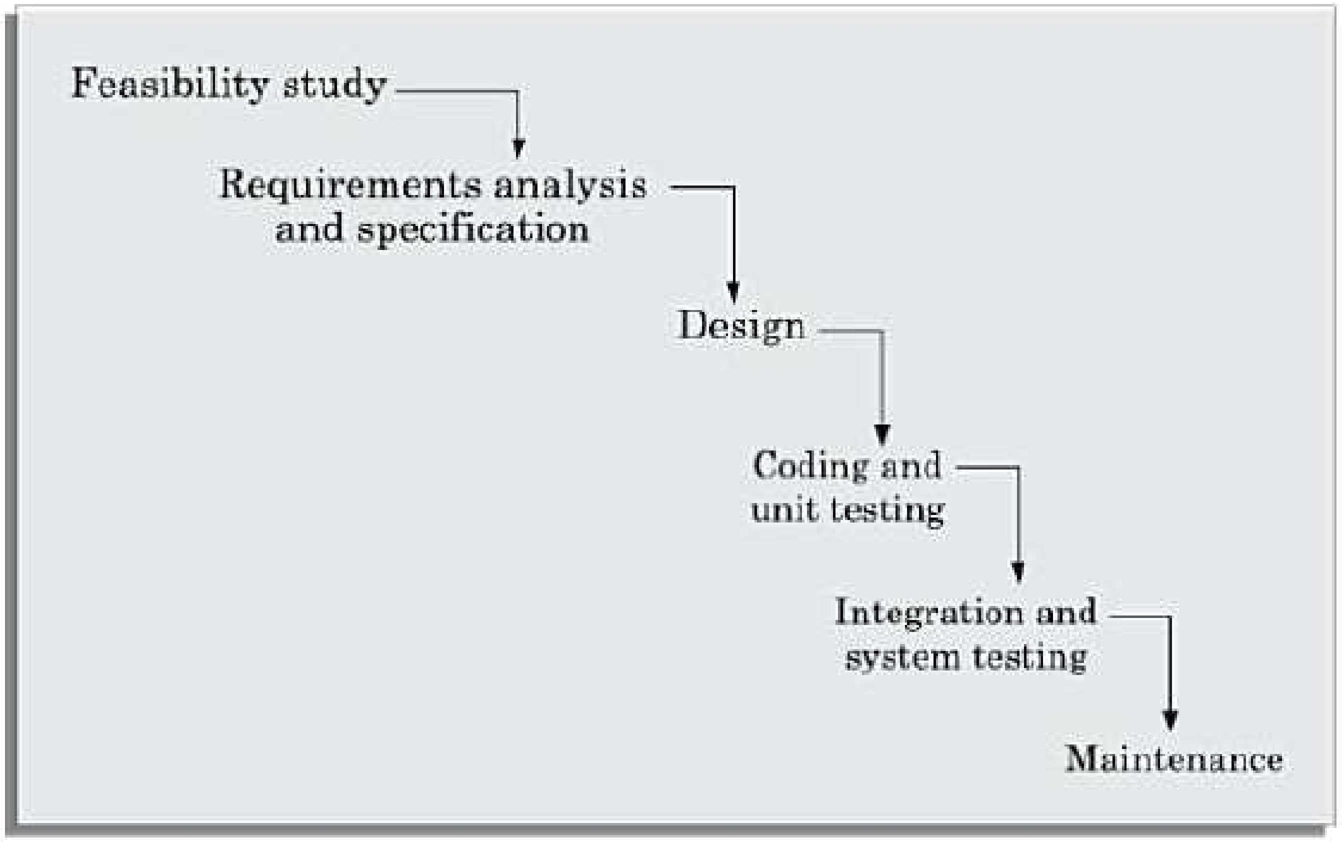
- **Software process models**

31.6.1 Linear sequential model

- The Linear sequential model suggests a systematic sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support.

- The waterfall model and its derivatives were extremely popular in the1970s and still are heavily being used across many development projects.

- **Classical Waterfall Model**


Feasibility study
Requirements analysis and specification
Design
Coding and unit testing
Integration and system testing
Maintenance

- **Classical Waterfall Model**

1. Feasibility study:

- The main focus of the feasibility study stage is to determine whether it would be financially and technically feasible to develop the software.

2. Requirements analysis and specification:

- The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly.

- Requirements gathering and analysis, and Requirements specification.

2.1 Requirements gathering and analysis:

- The goal of the requirements gathering activity is to collect all relevant information regarding the software.

2.2 Requirements specification:

- After the requirement gathering and analysis activities are complete, the identified requirements are documented. This is called a software requirements specification (SRS)document.

3. Design:

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.

3.1 Procedural design approach: Data-flow oriented approach

3.2 Object-oriented design approach:

4. Coding and Unit Testing

5. Integration and System Testing

- **α testing:**
- **β testing:**
- **Acceptance testing:**

6. Maintenance:

The total effort spent on maintenance of a typical software during its operation phase is much more than that required for developing the software itself.

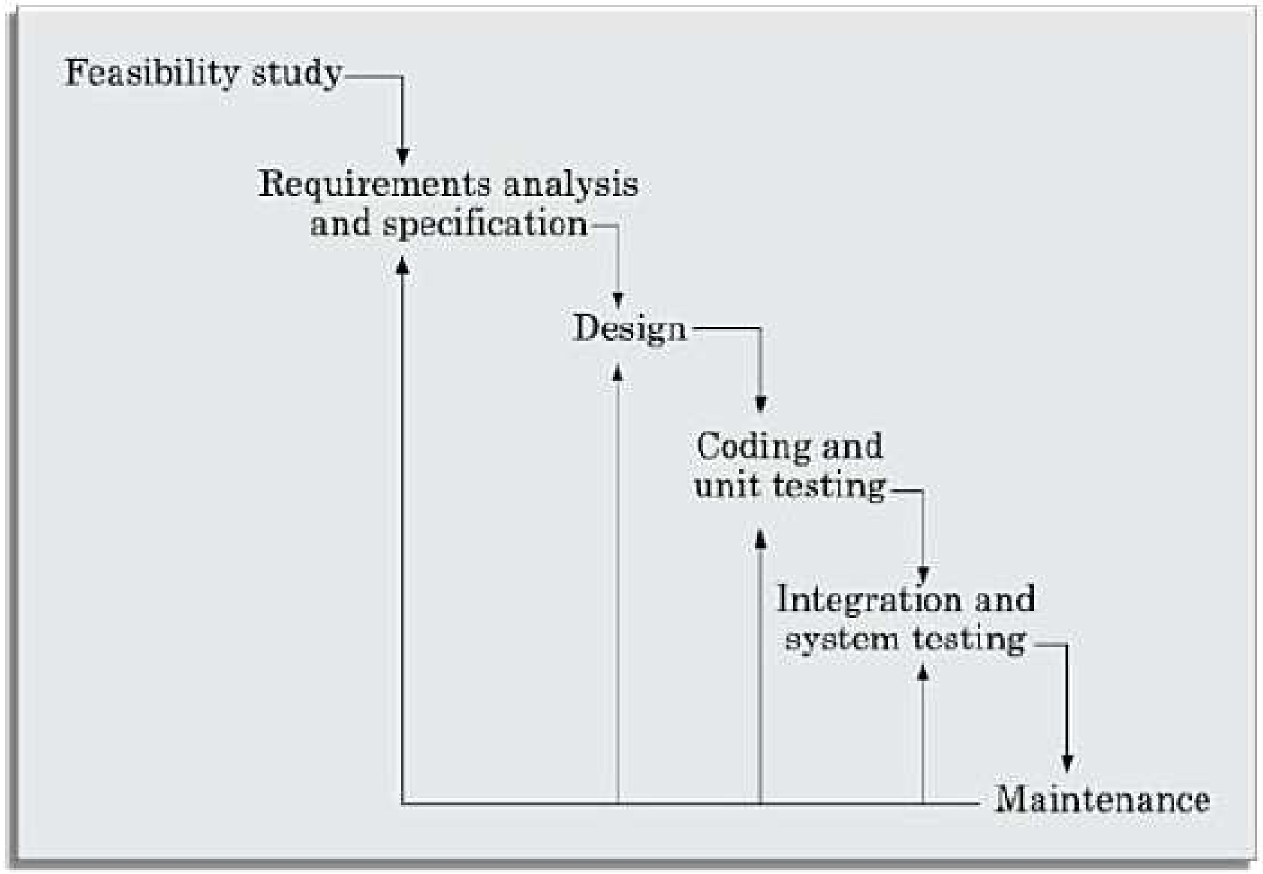
- **Corrective maintenance:**
- **Perfective maintenance:**
- **Adaptive maintenance:**

- **Verification vs Validation: Key Difference**

Verification	**Validation**
•The verifying process includes checking documents, design, code, and program	•It is a dynamic mechanism of testing and validating the actual product
•It does not involve executing the code	•It always involves executing the code
•Verification uses methods like reviews, walkthroughs, inspections, and desk- checking etc.	•It uses methods like Black Box Testing, White Box Testing, and non-functional testing
• Whether the software conforms to specification is checked	•It checks whether the software meets the requirements and expectations of a customer
•It comes before validation	•It comes after verification

- **Shortcomings of the classical waterfall model:**
  - No feedback paths:
  - Difficult to accommodate change requests:
  - Inefficient error corrections:
  - No overlapping of phases:

- **Iterative Waterfall Model:**


Feasibility study
Requirements analysis and specification
Design
Coding and unit testing
Integration and system testing
Maintenance

- **Shortcomings of the iterative waterfall model:**

The iterative waterfall model is a simple and intuitive software development model. It was used satisfactorily during 1970s and 1980s.

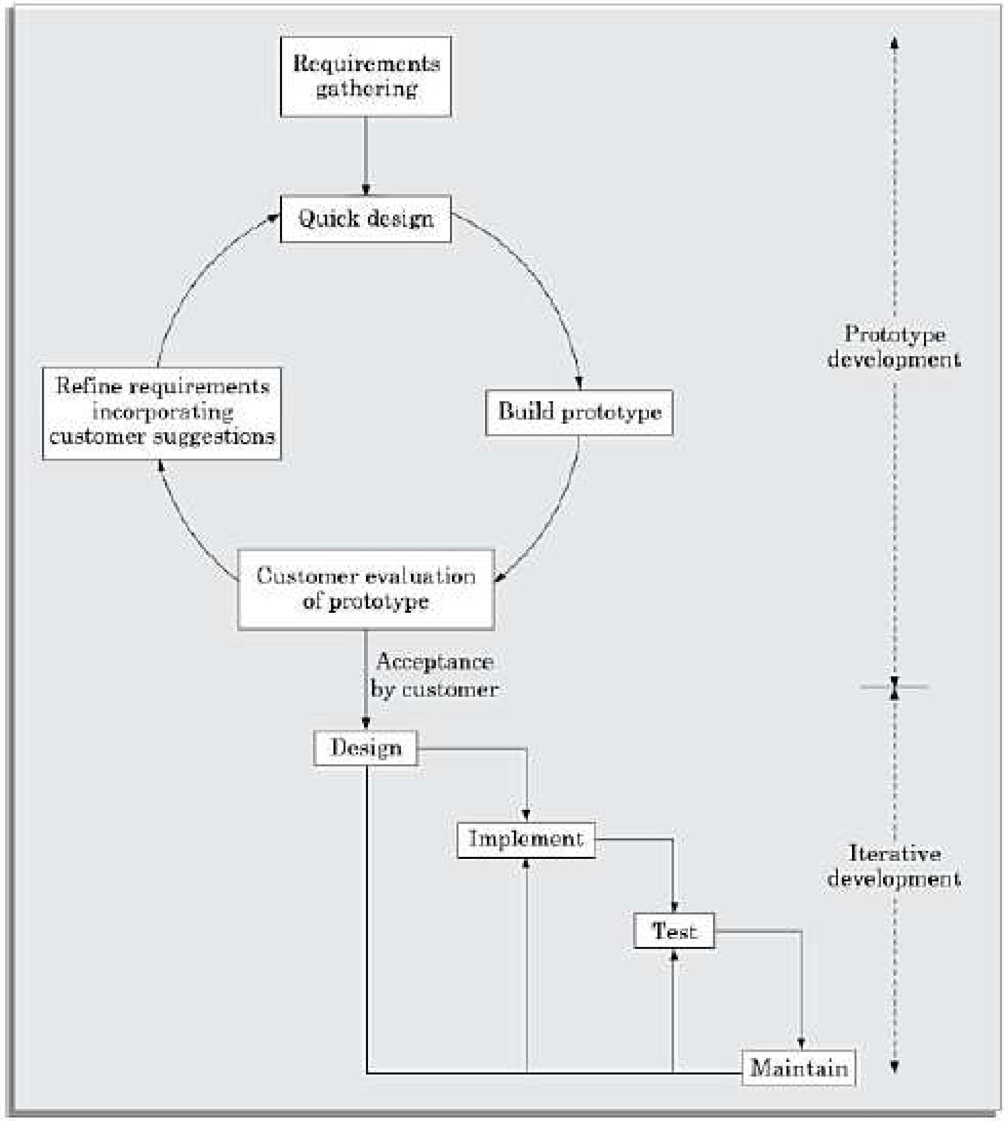
  - Difficult to accommodate change requests:
  - Incremental delivery not supported:
  - Phase overlap not supported:
  - Error correction unduly expensive:
  - Limited customer interactions:
  - Heavy weight:
  - No support for risk handling and code reuse:

- **Prototyping model:**

Software is developed through two major activities— prototype construction and iterative waterfall-based software development.

Prototype development:

Iterative development:


Requirements gathering
Quick design
Refine requirements incorporating customer suggestions
Build prototype
Customer evaluation of prototype
Acceptance by customer
Design
Implement
Test
Maintain
Prototype development
Iterative development

Advantages:

•This model is the most appropriate for projects that suffer from technical and requirements risks. A constructed prototype helps overcome these risks.

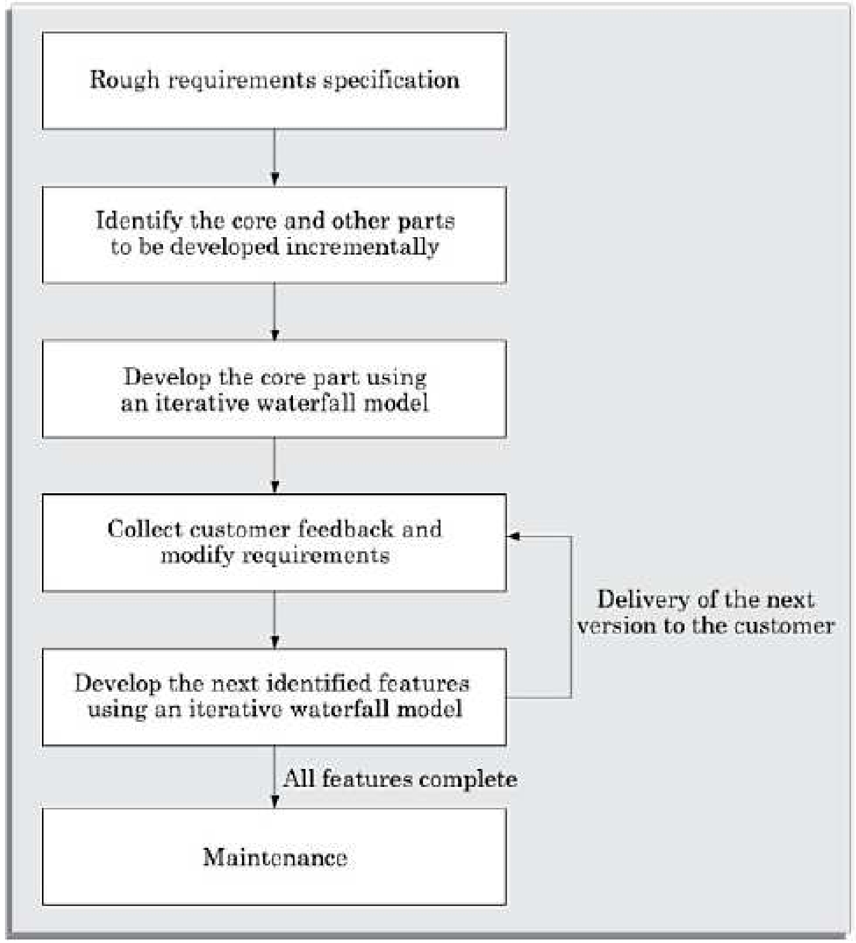
Disadvantages:

•The prototype model can increase the cost of development for projects that are routine development work and do not suffer from any significant risks.

•The prototyping model is effective only for those projects for which the risks can be identified upfront before the development starts.

- **Evolutionary model**

In Evolutionary model, the software is developed over a number of increments. At each increment, a concept (feature) is implemented and is deployed at the client site.


Rough requirements specification
Identify the core and other parts to be developed incrementally
Develop the core part using an iterative waterfall model
Collect customer feedback and modify requirements
Delivery of the next version to the customer
Develop the next identified features using an iterative waterfall model
All features complete
Maintenance

- **Advantages:**
  - Effective elicitation of actual customer requirements:
  - Easy handling change requests:
- **Disadvantages**
  - Feature division into incremental parts can be non-trivial:
  - Ad hoc design:

- **Spiral model**

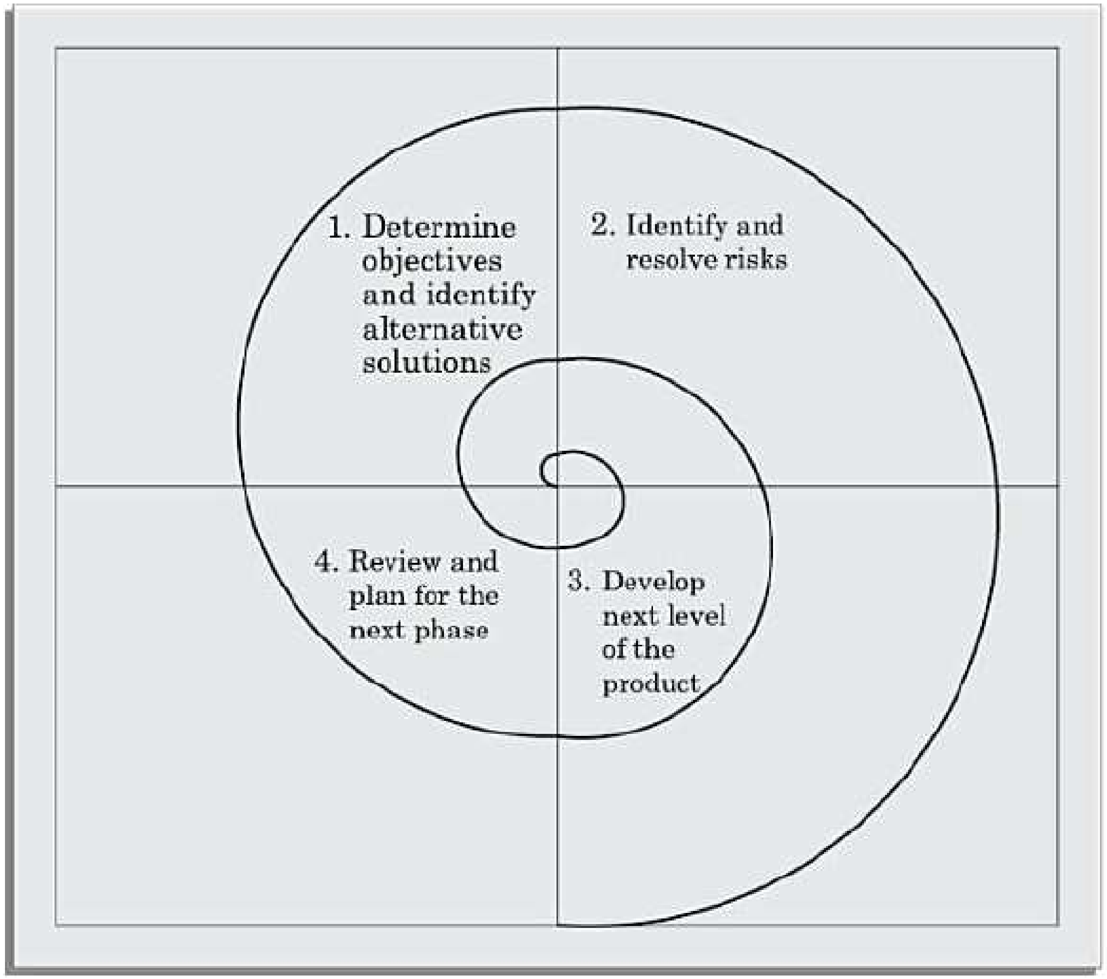
  - **Quadrant 1:** The objectives are investigated, elaborated, and analyzed. Based on this, the risks involved in meeting the phase objectives are identified. In this quadrant, alternative solutions possible for the phase under consideration are proposed.

  - **Quadrant 2:** During the second quadrant, the alternative solutions are evaluated to select the best possible solution. To be able to do this, the solutions are evaluated by developing an appropriate prototype.

  - **Quadrant 3:** Activities during the third quadrant consist of developing and verifying the next level of the software. At the end of the third quadrant, the identified features have been implemented and the next version of the software is available.

  - **Quadrant 4:** Activities during the fourth quadrant concern reviewing the results of the stages traversed so far (i.e. the developed version of the software) with the customer and planning the next iteration of the spiral.

31.6.4 Spiral model


1. Determine objectives and identify alternative solutions
2. Identify and resolve risks
3. Develop next level of the product
4. Review and plan for the next phase

- **Advantages:**
  - It is much more powerful than the prototyping model. Prototyping model can meaningfully be used when all the risks associated with a project are known beforehand.
  - All these risks are resolved by building a prototype before the actual software development starts.
- **Disadvantages:**
  - To the developers of a project, the spiral model usually appears as a complex model to follow, since it is risk-driven and is more complicated phase structure than the other models.
  - It would therefore be counterproductive to use, unless there are knowledgeable and experienced staff in the project.

- **RAD model:**

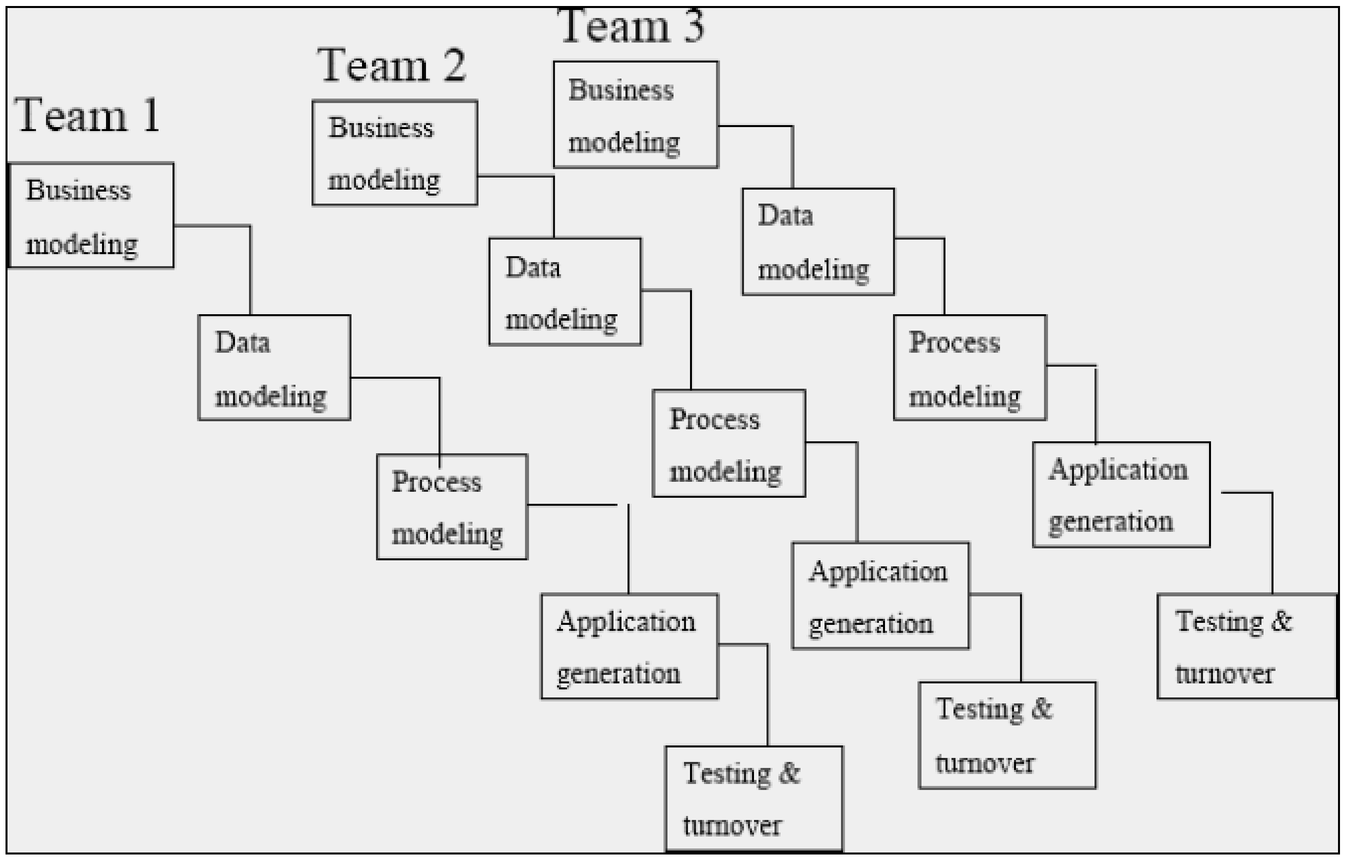
- Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping.

- A prototype is a working model that is functionally equivalent to a component of the product.

- In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

- **Phases of the RAD Model:**
    - **Business Modeling**
    - **Data Modeling**
    - **Process Modeling**
    - **Application Generation**
    - **Testing and Turnover**

- **Phases of the RAD Model:**



- **Advantages:**
  - Reduced development time.
  - Increases reusability of components
  - Quick initial reviews occur
  - Encourages customer feedback
  - Integration from very beginning solves a lot of integration issues.

- **Disadvantages:**
  - For large but scalable projects RAD requires sufficient human resources.
  - Projects fail if developers and customers are not committed in a much shortened time-frame.
  - Problematic if system cannot be modularized.
  - Not appropriate when technical risks are high (heavy use of new technology).

When to use RAD model?

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there’s high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

- ***Component based development:***

    - Commercial off-the-shelf (COTS) software components, developed by vendors who offer them as products, provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software that is to be built.

    - The component-based development model incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software.

The component-based development model incorporates the following steps:

- Available component-based products are researched and evaluated for the application domain in question.
- Component integration issues.
- A software architecture is designed to accommodate the components.
- Components are integrated into the architecture.
- Comprehensive testing is conducted to ensure proper functionality.
- The component-based development model leads to software reuse.

- **Fourth generation technique:**

1. The use of 4GT is a viable approach for many different application areas. Coupled with computer-aided software engineering tools and code generators, it offers a credible solution to many software problems.

2. Data collected from companies that use 4GT indicate that the time required to produce software is greatly reduced for small and intermediate applications and that the amount of design and analysis for small applications is also reduced.

- **Software process and project metrics: Software measurement:**
- **Terminologies:**

- **Measure**: Quantitative indication of the extent, amount, dimension, or size of some attribute of a product or process.
- **Metrics:** The degree to which a system, component, or process possesses a given attribute. Relates several measures (e.g. average number of errors found per person hour).
- **Indicators:** A combination of metrics that provides insight into the software process, project or product.
- **Direct Metrics:** Immediately measurable attributes (e.g. line of code, execution speed, defects reported).
- **Indirect Metrics:** Aspects that are not immediately quantifiable (e.g. functionality, quantity, reliability)
- **Faults:**
  - **Errors:** Faults found by the practitioners during software development
  - **Defects:** Faults found by the customers after release

- **Metric classification:**

- **Processes:**
  - Activities related to production of software

- **Products:**
  - Explicit results of software development activities.
  - Deliverables, documentation, by products

- **Project:**
  - Inputs into the software development activities
  - Hardware, knowledge, people

- **Process metrics:**

Process metrics are collected across all projects and over long periods of time to provide a set of process indicators that lead to long-term software process improvement.

Error Categorization and Analysis:

- All errors and defects are categorized by origin.
- The cost to correct each error and defect is recorded.
- The number of errors and defects in each category is computed.
- Data is analyzed to find categories that result in the highest cost to the organization.
- Plans are developed to modify the process

- **Project metrics:**

Project metrics enable a software project manager to assess the status of an ongoing project, track potential risks, uncover problem areas to control quality of software work products.

- **Product metrics:**

Product metrics focuses on the quality of deliverables. They are combined across several projects to produce process metrics.

- Measures of the Analysis Model
- Complexity of the Design Model
- Internal algorithmic complexity
- Architectural complexity
- Data flow complexity
- Code metrics

- **Software measurement:**
  - Size Oriented metrics
  - Function-Oriented metrics
  - Object-Oriented Metrics
  - Use-Case–Oriented

- **Function Point Metrics**

Function point metrics, measure functionality from the user’s point of view, that is, on the basis of what the user requests and receives in return.

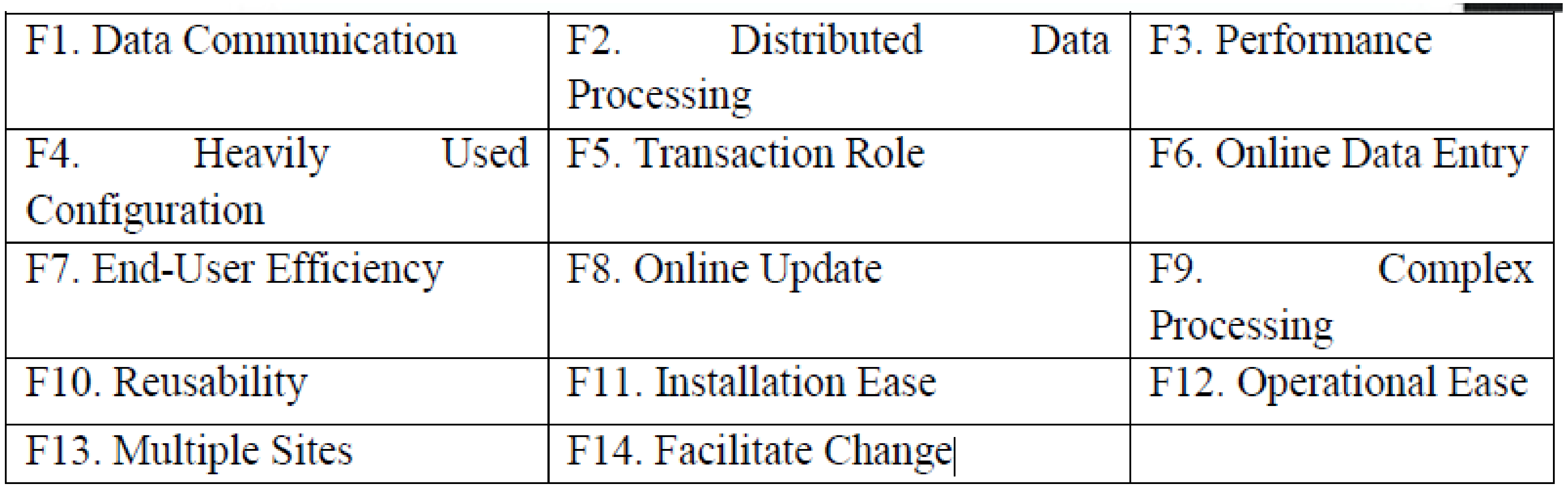
Information domain values are :

- Number of user inputs – Distinct input from user
- Number of user outputs – Reports, screens, error messages, etc.
- Number of user inquiries – On line input that generates some result
- Number of files – Logical file (database)
- Number of external interfaces – Data files/connections as interface to other systems.

- **Function Point Metrics**

Formula to count FP is FP = Total Count * [0.65 + 0.01*Σ(Fi)]

Where, *Total Count* is all the counts times a weighting factor that is determined for each organization. Fi (i=1 to 14) are complexity adjustment values.



F1. Data Communication	F2. Distributed Data Processing	F3. Performance
F4. Heavily Used Configuration	F5. Transaction Role	F6. Online Data Entry
F7. End-User Efficiency	F8. Online Update	F9. Complex Processing
F10. Reusability	F11. Installation Ease	F12. Operational Ease
F13. Multiple Sites	F14. Facilitate Change	

Thank You